Basic Calculus with the HP49G
Volume 1

By Nick Karagiaouroglou

Many, many special thanks to:

Thomas Rast	for his ideas about formulae layout and font usage for program listings and for the humour.
Veli-Pekka Nousiainen	for his ideas about key pressing conventions and also for the humour.

And to all guys out there who still keep on wanting the marathons after so many adventures.

Key pressing conventions

	Right shifted key
	Left shifted key
	Unshifted key
	Menu key (Soft key)
	Select the command from the command catalog of the HP49G or type it in the command line and enter it
	Alpha shifted key
	Press blue shift, hold it pressed, press F1, and then release both keys
	Press red shift, hold it pressed, press key 6, and then release both keys

Before you start working you should set your flags.
Enter the list { #A003008D8103F0h #0h #190101402000028h #0h } and press STOF

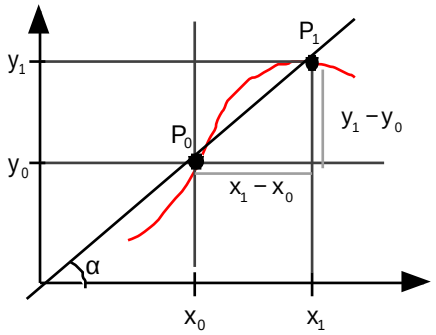
Basic Calculus with the HP49G - Volume 1 - Part 1

Hi again everybody!

This marathon comes indeed with a long delay. Many reasons have unfortunately made its completion a very hard job. It is as if problems suddenly decided to appear all together. The worst of them is that special kind of problem, because of which I exploded in the group. I can hardly keep myself calm and sit down and write, but it seems that mathematics has a calming influence on me. (Which can be used as evidence that I can't be normal. ;-))

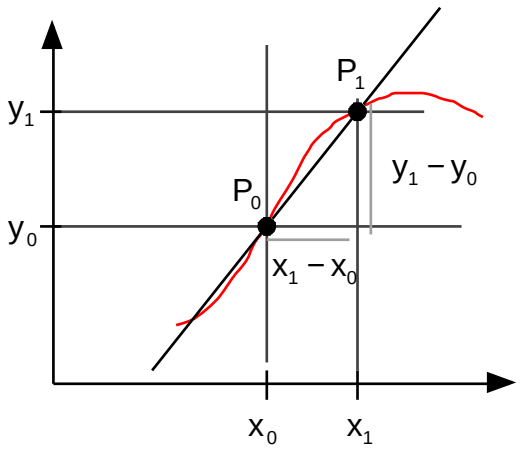
Anyway, before we start, some words and thoughts about mathematics and the world. After some thousands of years of history of humans, we have some real achievements and many mistakes. Between these achievements we have sciences and poetry, economic progress and technologies and many other things. But is there anything else between our achievements, that resembles mathematics in its search for the truth? Which of the human made achievements can be said to have the same validity as for example, that $A = A$, that something is identical to itself? (Might seem trivial, but its consequences are *huge*!) Even what is considered today to be the most beautiful poem, might be considered as trash after 1000 years. But could we ever say that $A \neq A$? And if we do, what next? It turns out that accepting the opposites of these basic axioms… makes everything possible. We *can* accept these opposites and try to build new mathematics out of them, but then each and every proposition will automatically be a theorem. (This is a proven fact.) And that is simply plain boring! If every statement is true, we have nothing interesting to search for. We find automatically all possible truth, because then everything is true. If we consider that we have great difficulties to even grasp what $A \neq A$ should mean, we have to start suspecting that our brain isn't made for boring things. That this built-in allergy against $A \neq A$ is somehow protecting us from getting stupid.
(Though we can get stupid in myriad other ways. ;-)) So, having this in mind, I ask again: Is there anything out there, with the same endurance like mathematics? Empires rise and fall, technologies come and go, religions appear and disappear, arts are in and then out. But these simple things, like $A = A$ remain. The tiny little letter, π, has a career of some thousands of years behind it, and it doesn't look as if its career would be approaching an end. It still is in your watch, in my Mac, in our houses, literally everywhere. If we let aside anything to which we hold because of belief, or tradition, or simply because we like it, and if we keep only what we can trust most, what remains then? Isn't that the simple crystal clear statements that built step by step the building of mathematics? There is something true, something almost godly in these statements. There is something about them, that makes it worthy for everybody to get interested and start thinking about them. Perhaps this is what made Paul Erdös say "he lives", when he meant "he does mathematics", and "he died", when he meant "he stopped doing mathematics". So, let's live again!

In the previous marathon, the Sequences, Series, and Limits Marathon, we had a real hard time. The HP49G doesn't provide many things for sequences and series, and so we had to program them ourselves. And in order to program them, we had to first take a look many things in the fields of sequences and series, which were of great importance and help. But because the HP49G does provide a vast amount of built-in features for calculus, this marathon will not be such a hard path. We will of course have to program now and then, but most of the time we will use the built-in features. There will be also a continuation of the calculus marathon in future, after we will have examined vectors and matrices. So this is going to be the Basic Calculus Marathon, or BACAMARA. (Sounds like a card game, but we are still not at games theory.)



We start with the picture of a plot of a function $y = f(x)$, some curve in general. If we consider the difference $\Delta x = x_1 - x_0 = h$, then the corresponding difference of the values of y is:

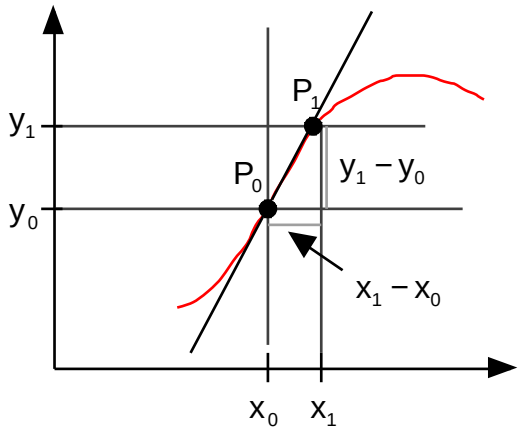
$$\Delta y = y_1 - y_0 = f(x_1) - f(x_0) = f(x_o + \Delta x) - f(x_0) = f(x_o + h) - f(x_0)$$



The line that goes through the points P_0 and P_1 is one secant of the curve. The slope of this line is:

$$\frac{\Delta y}{\Delta x} = \frac{f(x_0 + h) - f(x_0)}{h} = \tan(\alpha)$$

The ratio $\frac{\Delta y}{\Delta x}$ is the difference quotient. The nearer the point P_1 is to the point P_0, the better gets the description of the curve with the difference quotient at the point P_0. If we slip the point P_1 along the curve towards the point P_0, then the secant approaches a limit position, which is the tangent of the curve at the point P_0. The angle α goes towards a limit, the value ϕ_0, which is the angle between the x-Axis and the tangent. The quantity $\tan(\phi_0)$ is the slope of the tangent and the slope of the curve itself at the point P_0. That means, that when the point P_1 approaches the point P_0, the quotient $\frac{\Delta y}{\Delta x}$, approaches the slope of the curve at P_0. When the point P_1 approaches the point P_0, the difference Δx approaches 0. And so we have that the slope of the curve at P_0 is the limit of $\frac{\Delta y}{\Delta x}$ when $\Delta x \to 0$. We can find the slope of any curve by finding $\lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}$. This limit, *if it exists*, is the differential quotient or derivative of $y = f(x)$ at the point P_0 It is denoted as $f'(x_0)$, or $y'_{x=x_0}$, or $\left(\frac{dy}{dx}\right)_{x=x_0}$. From the last marathon we remember that the limit from the left has to be equal to the limit from the right. So, we have the definition of a function that has a derivative at a given point:



> **A function has a derivative at $x = x_0$, iff both the left and the right limits of the differential quotient exist and they are equal to each other.**

Another useful theorem for our purposes in this marathon is:

> **If a function has a derivative at $x = x_0$, then it is for sure continuous at this point.**

As we see, for a function to have a derivative at some point it is necessary but not sufficient to be continuous at that point. There are functions that are continuous at some point, but have no derivative there. There are even functions that are continuous in some interval of x-values and have no derivative at any point in this interval!

What does the HP49G provide considering all the above? Well, it doesn't have a built-in function for finding difference quotients. But why should it? Such things are piece of cake to program. A simple program for this is on the next page.

```
<<
  PUSH                          @Save current settings
   → f x x0 h                   @Store function, variable,
  <<                            @point x0 and difference of
                                @x-values
    f x x0 h + = SUBST          @Find f(x0 + h)
    f x x0 = SUBST              @Find f(x0)
    -                           @Find f(x0 + h) - f(x0)
    h /                         @Find (f(x0 + h) - f(x0))/h
    EXPAND                      @make it beautiful.
  >>
  POP                           @Restore saved settings.
>>
```

Store that in ΔQUOT. (The character Δ is ALPHA, →, F3.) Let's try some examples. Enter X^2-1, then X, then 1, and then $\frac{1}{10}$. Press the menu key ΔQUOT. The result is $\frac{21}{10}$. Press UNDO to undo this and get the previous stack. Drop the $\frac{1}{10}$, enter .1, and press ΔQUOT again to get 2.1, the numeric result for $\frac{21}{10}$. Notice that without putting the whole thing between the pair PUSH−POP, the second example would switch the calculator to approximate mode. Press again UNDO, drop the .1 and the 1, and enter X0 and h. If you now press ΔQUOT, you get the result 2 X0+h, the symbolic expression of the difference quotient. Of course we could use this result to find that $\lim_{h\to 0} 2\ X0+h = 2\ X0$. That is, we could use the difference quotient to find the derivative of X^2-1 at X_0. Having 2 X0+h on stack level 1, we could enter h=0 and then press lim to get 2 X0. But this isn't necessary, as the HP49G provides a big variety of commands to find derivatives.

We stay a bit more at ΔQUOT. One could think that in case of numeric calculation using the smallest possible numeric value for h would give the best possible approximation for the slope of the function at some point X_0. But there are problems. Let's try an example. We use ΔQUOT with decreasing values of h to calculate a good approximation of the slope of SIN(X) at X=π. Enter SIN(X), then X, then π, and then 1E−5. Now press ΔQUOT. The result is −.999999999984, quite near to the correct exact result −1. Let's do the same for h=1E−10. Press UNDO, drop the .00001 and enter 1E−10. Press again ΔQUOT. The result is now −.999999999996, even nearer to −1. Repeat for h=1E−12. Oops! Something went wrong here, because now the result is 0. How can this happen? Well, in the Sequences, Series and Limits Marathon we have examined many things about the limited precision and accuracy of the HP49G. What we see here is another result of these limitations. It helps to follow the program for understanding what happens. So, we will debug it using the built-in debugger of the HP49G. First of all, put all necessary arguments on the stack. They are SIN(X), X, π and 1E−12. Press → and then ΔQUOT to recall the program to the stack. Now, press PRG to go to the menu PRG. Press NXT twice and then RUN to go to the debugger menu. The first menu item from the left is DBUG. Press DBUG. This starts the program on stack level 1, but halts it immediately before the first command. (Note that the HLT annunciator at the top of the screen went on.) Now you can let the program run step for step. (This has nothing to do with the step by step feature of the CAS.) Press SST. This executes the next command in the program and halts the program again. Since the first command was PUSH, nothing changes on the stack. Note also that the command currently executed appears on the top of the screen. Press SST to execute the next command. On the top of the screen you see the local variables assignment: → f x x0 h. Apparently the whole sequence counts as a single command. Notice also how the arguments that stood on the stack, now are gone. They are consumed because they are used by the command →. Pressing SST again you see << t the top of the screen, which shows that we start with the local variables procedure. The next SST displays f on the top

of the screen. Since this is a local variable without quotes, its contents, SIN(X), are put on the stack. Continue pressing SST until the command / is executed. Now stack level 1 contains

$$\frac{\text{SIN}(\pi+.000000000001)-\text{SIN}(\pi)}{.000000000001}$$

We will need this expression again, so press ENTER to make a copy of it on stack level 2. Notice that we affect the stack used by the program, while the program is debugged. This is a very useful feature of the HP49G, as it allows to test and check many things while a program is single stepped. If you now press the menu key NEXT, the next two commands in the program are displayed on the top of the screen. These commands are EXPAND >>. Press SST. Here it happens! The returned result is 0.. Press CONT to let the program run to the end and quit. Note that the annunciator HLT is no more displayed. Why did the HP49G returned 0. when it expanded

$$\frac{\text{SIN}(\pi+.000000000001)-\text{SIN}(\pi)}{.000000000001}\ ?$$

Drop the 0. from stack level 1 and let's take a look at the above expression. As you can see the expression contains real numbers (numeric values). When we expand such expressions, the HP49G "sees" the numeric values and decides to switch to approximate mode. It uses then π's numeric approximation, 3.14159265359. The expression is turned to

$$\frac{\text{SIN}(3.14159265359+.000000000001)-\text{SIN}(3.14159265359)}{.000000000001}$$

Then the sum 3.14159265359 + .000000000001 is evaluated, which on an ideal machine with infinite precision should give 3.141592653591. But the HP49G has only 12 digits for real numbers. And so 3.14159265359 + .000000000001 returns… 3.14159265359! So the expression has been converted to

$$\frac{\text{SIN}(3.14159265359)-\text{SIN}(3.14159265359)}{.000000000001}$$

instead of

$$\frac{\text{SIN}(3.141592653591)-\text{SIN}(3.14159265359)}{.000000000001}$$

From this point on it is clear that further evaluation will return 0., since on the numerator of the quotient we subtract SIN(3.14159265359) from SIN(3.14159265359). That shows that we can't just make h as small as possible and expect to get the best possible numeric approximation for the slope of a function. But fortunately we don't need to bother which value of h is small enough for a good numeric result of the slope, but also big enough to avoid such problems. (So why does Nick write all this? He, he, just to have more stuff for the readers ;-))

And what happens if we don't use reals but exact numbers (integers)? Let's see. Enter again SIN(X), then X, then π, and then 1E−20. Now, enter XQ to convert 1E−20 to

$$\frac{1}{100000000000000000000}$$

Press ΔQUOT. Now the result is the expression:

$$100000000000000000000\ \text{SIN}\left(\frac{100000000000000000000\,\pi+1}{100000000000000000000}\right)$$

If you press →NUM at this point you will get −206761537357 ,

which means that the numeric problems strike back. In this case the problems come because of two reasons. First, the numeric value of

$$\frac{10000000000000000000\ \pi-1}{10000000000000000000}$$

is calculated as 3.14159265359, and this is only an approximation. The correct value should have been:

$$\frac{10000000000000000000\ \pi-1}{10000000000000000000}=\pi+\frac{1}{10000000000000000000}$$

That means that we should have at least a numeric approximation with 20 digits for π and in general for real numbers.

The second reason is that while the HP49G returns 0 if you expand $SIN(\pi)$, it returns $-2.06761537357E-13$ if you expand $SIN(3.14159265359)$. While π is exact, the number 3.14159265359 is not exactly π.

In this particular case you can press TEXPAND with the expression

$$10000000000000000000\ SIN\left(\frac{10000000000000000000\ \pi+1}{10000000000000000000}\right)$$

on stack level 1. This converts it to:

$$10000000000000000000\ \left(COS\left(\frac{1}{10000000000000000000}\right)0+SIN\left(\frac{1}{10000000000000000000}\right)-1\right)$$

Pressing →NUM now will return the correct result, -1.

All the above shows us that caution is needed when working with numeric expressions. (And that Nick will have enough stuff for all marathons ;-))

As already said, the HP49G provides a great amount of commands for finding derivatives and slopes. Let's take a look at them. We have the function ∂, which was also present at the HP48. This function has been modified a little bit, so it doesn't work exactly like in the HP48. In RPL syntax it takes the function from stack level 2, and the variable from stack level 1, and returns the derivative of the function for the specified variable. For example, enter $e^{SIN(W^2)}$, then enter W and then press ∂ to execute the function ∂. The result is $e^{SIN(W^2)}\ COS(W^2)\ 2\ W$. The same can be done using algebraic syntax. Go to the EQW and press ∂ to write the unfinished expression $\frac{\partial}{\partial\blacktriangleleft}(\blacksquare)$. The cursor blinks at the right of the ∂ in the "denominator" to indicate that the HP49G expects you to enter the variable of derivation. Enter W. Press ▶ to go inside the parentheses and enter $e^{SIN(W^2)}$. Now you have:

$$\frac{\partial}{\partial W}\left(e^{SIN(W^2)}\right)$$

Press ENTER to put that on stack level 1. Before we go further, let's see how the expression looks like when it isn't shown in pretty print. Press ↩ and then ▼ to edit the expression not in the EQW but in the command line, where no pretty print is used. Now you see: '$\partial W(EXP(SIN(W^2)))$', which contains only one ∂. The general syntax of derivatives in the command line is '$\partial var(function)$', where var stands for the variable of derivation and $function$ for the function whose derivative you want to find.

Press now ENTER to put the expression back to the stack. If you expand

it, then you get the same result as before, $e^{SIN(W^2)}\ COS(W^2)\ 2\ W$.

Now we can see how we can find the slope of a function at a given point. We can find the derivative and substitute the value of the variable at that point. Our example from the previous page was to find the slope of $SIN(X)$ at $X = \pi$. We enter $SIN(X)$, then X, and then we use ∂. The result is $COS(X)$. Now we enter $X = \pi$, we press SUBST and then EXPAND, and we get -1. Or at the point where the HP49G returns the result of the derivation $COS(X)$, we enter the list $\{X\}$, then we enter π, we press + to create the list $\{X\ \pi\}$ and then we press ▶ to use the function | (where). The result $COS(\pi)$ can be expanded then to -1. In this case we didn't enter the list $\{X\ \pi\}$ directly because this would create a list which contains the *function* π and not the algebraic object π. If you enter $\{\pi\}$, press HEAD and then TYPE, you see that its object type is 18. (function). But if you enter π alone and press TYPE, then the result is 9. (algebraic object). That means that π is a function which puts the algebraic object π on the stack. Since the function | doesn't work with arguments of type function, it would error out.

We can use the function | in an easier way that also looks better. Go to the EQW and enter:

$$\frac{\partial}{\partial X}(SIN(X))$$

Select the whole expression and press ▶ to write | to the right of the expression. Complete the expression to

$$\frac{\partial}{\partial X}(SIN(X))\Big|_{X=\pi}$$

Press ENTER to put the expression on the stack, and expand it to -1.

Another way to find a slope, inherited from the HP48, is to find the derivative for some variable, in which we have stored a value. For example, store π in variable T. Go to the EQW and enter:

$$\frac{\partial}{\partial T}(SIN(T))$$

Put that on the stack and expand. The result is -1. The HP49G found the derivative of $SIN(T)$ to be $COS(T)$, and then proceeded using the value π stored in T, found $COS(\pi)$, and expanded that to -1. But note: If the variable is the current VX, then we get problems. Suppose the current VX is X. If you store π in X, enter

$$\frac{\partial}{\partial X}(SIN(X))$$

and then expand, you get the question to purge the current variable. If you choose "No", then the operation errors out with Mode Switch Cancelled". If you choose "Yes", then X is purged and the calculation returns $COS(X)$, because the variable X doesn't exist any more. In this case instead of expanding you can press EVAL twice, to avoid the question about purging the current variable.

We continue examining the behaviour of ∂. A very interesting question is, what happens when we take derivatives of expressions with variables that themselves contain other expressions. Since there is a big number of cases and sub cases, we try to go as systematically as we can. (Or rather, as systematically as Nick can imagine ;-)) First we are going to store some expressions in some variables.

Store X^2 in R.

Store $SIN(Y)$ in S.

Store $S^2 - R$ in U.

We also need some user defined functions.

In the EQW enter:

$$V(X) = \frac{X^3}{X-1}$$

Put the equation in the stack and press DEF to create the user defined function. The same way create the user functions:

$$W(R) = R\ (R+1)$$

$$Z(R,X) = \frac{R-1}{X}$$

$$Q(S) = \frac{S}{R}$$

Last thing, we create user functions with RPL syntax. Enter:

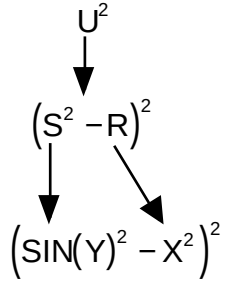
```
<< → X
  <<  0 1 3 FOR I
        X I R→I ^ +
      NEXT
  >>
>>
```

Store it in Z1. Now enter the program:

```
<< → R X
  <<  R EXP S EXP + X / >>
>>
```

Store it in Z2.

Now we are ready to start examining what ∂ does, when applied to different combinations of functions and expressions with variables that contain other expressions.



Enter 'R' (with quotes) and press ENTER to make copy it on stack level 2. Press ∂ to take the derivative of R for R. The result is 1, which shows that ∂ didn't care about the fact that X^2 is stored in R. If you enter 'R' and then X, and take the derivative, then you get a fat 0, which again shows: When some expression is on stack level 2 and we take its derivative for some of its variables using the function ∂, then derivation is carried out without first evaluating that variable. This is good for finding slopes of functions by storing something in the variable for which we take the derivative and then using ∂, as we already saw on the previous page. Another example of this case: Enter S^2, then 'S' (with quotes) and press ∂ to get 2 S. The same will happen if you enter

$$\frac{\partial}{\partial S}\left(S^2\right)$$

from the EQW and press EVAL. If you expand instead of evaluating, then not only the derivative 2 S will be returned, but also evaluation of S will be carried out *after derivation*, and the result will be 2 SIN(Y).

If you want to first evaluate some variable contained in an expression, and then take the derivative, you have to press EVAL first. For example, enter U^2. We stored $S^2 - R$ in U, SIN(Y) in S, and X^2 in R. If you want to take the derivative of the evaluated expression U^2 for X, then press EVAL first. This completely evaluates U^2 to $SIN(Y)^4 - 2\ X\ SIN(Y)^2 + X^4$. Now you can enter X, press ∂ and then expand, to get $-\left(4\ X\ SIN(Y)^2 - 4\ X^3\right)$.

A problem that we have is, how could we for example take the derivative of the *partially* evaluated U^2 for S? If we evaluate U^2, evaluation goes all the way down and doesn't stop at any intermediate step. That's what the command SHOW is for. If you enter U^2, then 'S' (in quotes) and press SHOW, then evaluation will stop when the variable S is shown. This will result in $\left(S^2-R\right)^2$, allowing us to enter 'S' and take the derivative, by pressing ∂.

Until now we have seen that ∂ doesn't evaluate the variables of expressions. However, ∂ shows a special behaviour when the expression, which we take the derivative of, is a user defined function. Let's take a look at that. Enter $V(X)^2$ and then X. If you now press ∂ then the derivation takes a bit longer, and returns:

$$2\ \frac{X^3}{X-1}\ \frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}$$

This shows: Since V is a user defined function, $V(X)^2$ is *first* evaluated to:

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

and *then* derivation for X is carried out! For the function ∂, the user defined functions are much like the built-in functions. Exactly the same result will be returned if you enter

$$\frac{\partial}{\partial X}\left(V(X)^2\right)$$

and then evaluate. If you enter

$$\frac{\partial}{\partial X}\left(V(X)^2\right)$$

and expand, then you get the result:

$$\frac{\left(4\ X^3-6\ X^2\right)\ \frac{X^3}{X-1}}{X^2-2\ X+1}$$

This is a bit strange, since it is indeed correct, but *not completely expanded*. You can press EXPAND once again, to get the completely expanded form. If you enter

$$\frac{\partial}{\partial X}\left(V(X)^2\right)$$

and then press FACTOR or COLLECT, nothing happens and the expression remains unchanged on stack level 1. This might seem not good, but it allows to collect unevaluated differential forms, which for example can be used to bring differential equations to a much more readable form. For example, enter

$$\frac{\partial}{\partial X}\left(V(X)^2\right)\ X+\frac{\partial}{\partial X}\left(V(X)^2\right)\ A$$

and press COLLECT, to get

$$(X+A)\ \frac{\partial}{\partial X}\left(V(X)^2\right)$$

If you want the opposite to happen, then don't press EVAL or EXPAND, but press DISTRIB. (DISTRIB is the first menu item of menu CONVERT/REWRITE. If you press DISTRIB with the last result still on stack level 1, then you will get

$$X\ \frac{\partial}{\partial X}\left(V(X)^2\right)+A\ \frac{\partial}{\partial X}\left(V(X)^2\right)$$

Is there any way to transform the sub expression

$$\frac{\partial}{\partial X}\left(V(X)^2\right)$$

of the above expression to

$$2\ V(X)\ \frac{\partial}{\partial X}\left(V(X)\right)$$

getting thus a form that contains only differential forms of the function $V(X)$, but not of $V(X)^2$? Yes, there is. Leave the last result on stack and press MODE. Then press the menu key CAS, and activate the option _Step/ Step. Press OK twice to return to the stack. Press ENTER to copy

$$X\ \frac{\partial}{\partial X}\left(V(X)^2\right)+A\ \frac{\partial}{\partial X}\left(V(X)^2\right)$$

to stack level 2. If you evaluate, then you get

$$(2\ A+2\ X)\ V(X)\ \frac{\partial}{\partial X}\left(V(X)\right)$$

Press ▶ to swap stack levels 1 and 2. If you now expand, then you get the result:

$$(2\ A+2\ X)\ \frac{X^3}{X-1}\ \frac{(X-1)\ \frac{\partial}{\partial X}\left(X^3\right)-X^3\ \frac{\partial}{\partial X}(X-1)}{SQ(X-1)}$$

This is the same like the result on stack level 2, but the user defined function $V(X)$ has been additionally evaluated and the derivative

$$\frac{\partial}{\partial X}\left(\frac{X^3}{X-1}\right)$$

has been rewritten as

$$\frac{(X-1)\ \frac{\partial}{\partial X}\left(X^3\right)-X^3\ \frac{\partial}{\partial X}(X-1)}{SQ(X-1)}$$

according to the rules for differentiating ratios. Now, deactivate the step by step feature. We will see more about this feature later on.

The next question is: what happens when we take the derivative of some user defined function, to which we have given a variable as argument, that itself contains some expression? Enter the expression $V(R)$. Now, enter 'R' and press ∂. The result is… 0! Perhaps it seems a bit strange, because the expression $V(R)$ does contain R and taking the derivative for R, we would expect some result different than 0. But it is completely understandable, if we think again about the special behaviour of ∂ when we take derivatives of user defined functions. In this case the user defined function *is evaluated before derivation*. This means that $V(R)$ was first evaluated to

$$\frac{R^3}{R-1}$$

Then, R was evaluated, which returned

$$\frac{X^6}{X^2-1}$$

since R contains X^2. It was this expression of which the derivative for R was taken. And since the expression

$$\frac{X^6}{X^2-1}$$

doesn't depend on R, the derivative was 0. Enter V(R) again, then enter X and press ∂ to get the result

$$\frac{(X^2-1)\ 6\ X^5-X^6\ 2\ X}{SQ(X^2-1)}$$

which is the derivative of the completely evaluated user defined function V(R) for variable X.

Things go different if you use algebraic syntax. Enter

$$\frac{\partial}{\partial R}(V(R))$$

If you evaluate this, then the result is:

$$\frac{(R-1)\ 3\ R^2-R^3}{SQ(R-1)}$$

which is the derivative of the user defined function V(R), where the argument R hasn't been evaluated to X^2. But let's be organised and get a systematical overview.

The problem that arises is, how to let V(R) be evaluated up to

$$\frac{R^3}{R-1}$$

and then take the derivative, without replacing R by its contents first? Well, here comes a hidden super command, the command QUOTE, which is perhaps one of the most underestimated commands of the HP49G. QUOTE is much like putting variables in expressions in an additional pair of single quotes, giving us the power to control exactly what is evaluated. You know of course that for example entering R will put X^2 on the stack, because X^2 is stored in R. But entering 'R' will simply put 'R' on the stack without evaluating the variable. We extend this concept for algebraic expressions. Go to the EQW and enter V(QUOTE(R)). This is much like as if we had entered V('R'). Press ENTER a couple of times, because we will need several copies of this expression. If you now press EVAL, then the result will not be

$$\frac{X^6}{X^2-1}$$

but

$$\frac{R^3}{R-1}$$

When we evaluate V(R), all possible evaluations are carried out before the user defined function is evaluated itself. That means, that R itself is evaluated, and since it contains X^2 the user defined function V is given the argument X^2. But when we evaluate V(QUOTE(R)), it is as if we were evaluating V('R'), and the quotes around R prevent it from being evaluated, thus giving R and not X^2 to the user defined function V. Press ◀ to drop the result

$$\frac{R^3}{R-1}$$

and bring V(QUOTE(R)) on stack level 1. Enter 'R' (in quotes) and

press ∂ to get

$$\frac{(R-1)\ 3\ R^2-R^3}{SQ(R-1)}$$

This is the derivative of the user defined function V(R) evaluated up to:

$$\frac{R^3}{R-1}$$

The more demanding people will say now, "Yes, but the result is not completely expanded and if I expand now, I will get again an expression containing X s." And here we have the whole glory of QUOTE. You can even quote more than once. Enter V(QUOTE(QUOTE(R))), which is much like V(''R''). Enter 'R' and press ∂ to get:

$$\frac{(QUOTE(R)-1)\cdot 3\cdot QUOTE(R)^2\cdot d1QUOTE(R)-QUOTE(R)^3\cdot d1QUOTE(R)}{SQ(QUOTE(R)-1)}$$

If you evaluate now, you will get

$$\frac{2\ R^3-3\ R^2}{R^2-2\ R+1}$$

which is the fully expanded result that still contains R s and not X s!! Superb, isn't it? The first QUOTE prevented variable R from being evaluated when we took the derivative. The second prevented it from being evaluated when with pressed EVAL. The sub expression d1QUOTE(R) might puzzle you, so here is a small explanation. This expression means the derivative of the function QUOTE for its first argument, R. When the HP49G has to take a derivative from a function of which it doesn't know what the derivative is, it returns such expressions. Since the function QUOTE simply returns its argument quoted, the expression d1QUOTE(R) is the same as:

$$\frac{\partial}{\partial R}('R')$$

which the next evaluation turned to 1, leaving the final result correct. The bigger explanation about such expressions will be given later on. Note also that in the above examples we used EVAL rather than EXPAND, because the latter would have expanded the expression completely. For example, if you enter V(QUOTE(R)) and expand, then the user defined function V is given the argument R and the intermediate result is:

$$\frac{R^3}{R-1}$$

But EXPAND goes further after this, finds out that R contains X^2 and evaluates R, giving you the final result,

$$\frac{(X^2)^3}{X^2-1}$$

If you want to enter the derivative for R in algebraic syntax, then you don't need to quote. You simply enter

$$\frac{\partial}{\partial R}(V(R))$$

If you evaluate this, then the result is:

$$\frac{(R-1)\ 3\ R^2-R^3}{SQ(R-1)}$$

the same like using QUOTE once in RPL syntax. For being able to completely expand afterwards without evaluating R, you enter:

$$\frac{\partial}{\partial R}\left(V(\text{QUOTE}(R))\right)$$

and evaluate twice to get:

$$\frac{2\ R^3 - 3\ R^2}{R^2 - 2\ R + 1}$$

which is the same result like using QUOTE twice in RPL syntax.

And further we go: What happens when a user defined function contains a variable in which we have stored some expression containing other variables? Take for example the user defined function $W(R) = R\ (R+1)$. Its argument, R, is also a global variable which contains X^2. How are such things evaluated? Here we have to always remember first: The local variables are completely different entities from those that exist globally. As long as the user defined function is evaluated, all evaluations of local variables do nothing more than simply putting the contents of the local variables on the stack. The user defined function W in RPL syntax would look like:

```
<< → R           @Store argument in local R
                 @which doesn't have to do
                 @anything with the existing
                 @global R, that contains X^2
  << R           @Put contents of local R on
     R           @the stack twice. Don't evaluate
                 @these contents further.
     1           @Enter 1.
     +           @Calculate R+1
     *           @Calculate R*(R+1)
  >>
>>
```

If you enter X and press the menu key W, then X is stored locally in R. This local R exists only while the algebraic $R\ (R+1)$ is evaluated. It disappears afterwards. The variable R in the defining procedure $R\ (R+1)$ is *not* the global variable R which contains X^2. The result is $X\ (X+1)$ and not $X^2\ (X^2+1)$.

The question is, what happens if we give 'R' as argument for W? Well, then the global name R is stored in the local name R. Throughout evaluation of $R\ (R+1)$, the contents of the local R are put on the stack, but not evaluated any further! Since those contents are the global R, it is the global R that is put on the stack and doesn't get evaluated. When the user defined function finishes, it leaves $R\ (R+1)$ on the stack, but now R is the global name. Now we can evaluate or expand $R\ (R+1)$, to let the global variable R be also evaluated and replaced by X^2.

If you on the other hand enter $W(R)$ and evaluate or expand, then the evaluation will return $R\ (R+1)$, R being the global variable as above. But this will be further evaluated, replacing R with its contents X^2. The overall evaluation will return $X^2\ (X^2+1)$. Notice the difference to RPL syntax. Why is there a difference? Well, this is completely logical. In RPL syntax, you enter 'R' and apply the user function W on that argument. Since all contents of the local variable are simply put on the stack, the result is $R\ (R+1)$. But when you explicitly evaluate $W(R)$, this *includes* EVALuation of the result $R\ (R+1)$, which results in $X^2\ (X^2+1)$. Evaluating $W(R)$ is meant inclusively for the global variable R that is contained in the result $R\ (R+1)$. The important thing to note here is that it is not the local variable R that returns X^2 in any way. To understand this better, take a look at the following user defined function (next page):

```
<< → R            @Store argument in local R
                  @which doesn't have to do
                  @anything with the existing
                  @global R, that contains X^2
  << R            @Put contents of local R on
     EVAL         @stack and evaluate.
     R            @Contents of local R on stack
                  @these contents further.
    1             @Enter 1.
    +             @Calculate R+1
    *             @Calculate R*(R+1)
  >>
>>
```

If you enter 'R' and let this program run, then the following happens: First, the global name R is stored in the local name R. Then the contents of the local name R are put on the stack. This puts 'R' on the stack. Then we explicitly evaluate the object on stack level 1, which is the global 'R'. This puts X^2 on stack level 1. Then once again, the contents of the local name R are put on the stack. This puts 'R' on the stack once again. We add 1, and so we get $R+1$. Then we multiply and thus we get $X^2\ (R+1)$ as result.

Now what happens if we take derivatives of such functions? Here the rule that we already know also applies. The user defined function gets evaluated *first*, and then derivation follows. Enter $W(X)$ and then X. Press ∂ to get the result $X+1+X$. The user defined function $W(X)$ was first evaluated, and that returned $X\ (X+1)$. Then the derivative for X was found, $X+1+X$. (If you enter X and press W, then the result is $X\ (X+1)$. If you enter X again and press ∂ then you get the same result, $X+1+X$.) Exactly the same if you enter

$$\frac{\partial}{\partial X}(W(X))$$

and press EVAL. If you press EXPAND instead of EVAL, then the result will be $2\ X+1$, that is completely expanded.

Now the tricky part. If you enter $W(R)$, then 'R' and then press ∂, the result is… 0! And it complies to the rule: The user function $W(R)$ is evaluated first. We already said that evaluating $W(R)$ (algebraic syntax) goes all the way down and returns $X^2\ (X^2+1)$. If we take the derivative of this expression for R, we get 0, because the expression doesn't contain R. The same happens if you expand or evaluate the expression

$$\frac{\partial}{\partial R}(W(R))$$

But if you enter 'R', and then press W, then enter 'R' again and then press ∂, the result will be $R+1+R$. As we already saw, in RPL syntax no evaluation of the global variable R took place. So, we get the result $R\ (R+1)$ when we apply the user defined function W on the argument 'R' which is on the stack. After this there is no user defined function that has to be evaluated before taking the derivative. The expression $R+1+R$ doesn't contain any user defined function, and so ∂ simply takes the derivative of this expression.

The technique using QUOTE can also be used here. If you enter $W(\text{QUOTE}(R))$, then 'R' and then press ∂, the result will be $R+1+R$ and not 0. In algebraic syntax, if you enter

$$\frac{\partial}{\partial R}(W(\text{QUOTE}(R)))$$

and press EVAL, the HP49G returns the result:

$$\text{d1QUOTE}(R)\ (\text{QUOTE}(R)+1)+\text{QUOTE}(R)\ \text{d1QUOTE}(R)$$

If you evaluate this again, you get $2\ R+1$.

If you enter $W(R)$, then X, and then press ∂, then the result is $2\ X\ (X^2+1)+X^2\ 2\ X$. This shows again, the evaluation of $W(R)$ took place, the way we already know, which returned $X^2\ (X^2+1)$, and then the derivative for X of this result was found. The strange thing comes now. I would expect that evaluating

$$\frac{\partial}{\partial X}(W(R))$$

wouldn't return 0. But it does! In this case the uniformity of the behaviour breaks down. Because if the user defined function $W(R)$ would be first evaluated following the rules on the previous page, then the result of this evaluation should have been $X^2\ (X^2+1)$. Taking the derivative afterwards should return $2\ X\ (X^2+1)+X^2\ 2\ X$ and not 0. So here we have a problem because the rules are broken. The conditions that must be true in order for this problem to appear are:

1) We have a user defined function that uses local names which also exist globally.
2) We give that user function the global name as argument, which it also uses itself locally.
3) We use ∂ in algebraic syntax.

I would be very glad if someone could put some light in this mystery, so if somebody out there starts experimenting, then please post your results and tell us more about this question.

For user defined functions that use both variables in which we didn't store anything and variables that do contain something, the same rules apply in combination. Try some examples for yourself using the user defined function Z with different combinations of arguments.

Taking the derivatives of user defined functions follows the same rules, also when the functions are nested. For example, enter $W(V(X))$, enter X and then press ∂ to get the expression:

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

How is this result produced? Let's follow what the HP49G does. First the most inner function, $V(X)$, was evaluated, and returned:

$$\frac{X^3}{X-1}$$

which was used as argument for the function W. The calculator has evaluated

$$W\left(\frac{X^3}{X-1}\right)$$

This produced

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

This result was then differentiated for X and produced the final result. Exactly the same would happen, if we entered

$$\frac{\partial}{\partial X}(W(V(X)))$$

and evaluated.

Now we do the same using variable R as argument. If you enter 'R',

then press V, then press W, enter 'R' again, and press ∂, you find:

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

If you enter W(V(R)), then 'R' and then press ∂, then you find 0. In the first case, the HP49G used the argument which we gave it, namely 'R' and just applied the user functions V and W on this argument. Since the result didn't contain any user defined functions any more, the function ∂ just differentiated what it found on the stack, without any evaluation. In the second case however, the function ∂ found an expression that contained user defined functions, so it evaluated them first, found

$$\frac{X^6}{X^2-1}\left(\frac{X^6}{X^2-1}+1\right)$$

and so the differentiation for R returned 0.

Until now we used user defined functions with an algebraic definition. But we can also use RPL definitions that return an algebraic object. For example enter

$$\frac{\partial}{\partial X}(Z1(X))$$

and expand to get $3\ X^2+2\ X+1$. Alternatively you can also enter Z1(X), then X, and then press ∂ to find the derivative.

Using the same thoughts like before, we can predict what will happen if we evaluate

$$\frac{\partial}{\partial R}(Z1(R))$$

Since Z1 has to be evaluated first, the following happens:

```
<< → X           @Store global 'R' in local X
  << 0           @Enter 0
     1 3 FOR I   @Do with I from 1 to 3
      X          @Put contents of local X on
                 @the stack. This puts 'R' on
                 @the stack.
      I R→I      @Put I on the stack, make it integer
      ^          @Find 'R^I'
      +          @Add to the 0 that we entered at the
                 @start of the program. (The next
                 @times 'R^I' will be added to the sum
                 @that is already on the stack.)
     NEXT        @Increment I, do again
  >>
>>
```

In all the above events, R never gets evaluated and so the result of this function is $R+R^2+R^3$, and not $X^2+\left(X^2\right)^2+\left(X^2\right)^3$. That means, the next thing that happens, taking the derivative for R, will return $1+2\ R+3\ R^2$. Enter

$$\frac{\partial}{\partial R}(Z1(R))$$

and evaluate to see for yourself. Of course, if you enter

$$\frac{\partial}{\partial R}(Z1(R))$$

and expand instead of evaluating, then after the result $1+2\ R+3\ R^2$ has been found, the HP49G doesn't stop but proceeds, replacing R

with its contents, which are X^2 (and doing some reordering). So the result of expanding is $3\ (X^2)^2 + 2\ X^2 + 1$.

On the other hand, if we enter $Z1(R)$, then 'R' and press ∂ to find the derivative, the result is 0. Evaluate $Z1(R)$ to understand why. In this case R itself is evaluated before $Z1$. What will be returned if you take the derivative of $Z1(QUOTE(R))$ for 'R'?

The function ∂ can also be used for carrying out formal derivations. For example, enter $F(X)$ and then X, and then press ∂ to get $d1F(X)$. This result denotes the derivative of $F(X)$ for the first variable, which is X. Enter $F(X,Y)$ and then Y and press ∂ again to get $d2F(X,Y)$. This means the derivative of $F(X,Y)$ for the second variable, which is Y. For now, we only note that such formal derivations are "not clear to the users that the great makers[1] left uninformed". If you enter $F(X(T))$ then T and then press ∂, the result is $d1X(T)\ d1F(X(T))$. The result means the product of the derivative of the "inner" function $X(T)$ for its first variable T, and the first derivative of the function $F(X(T))$ for its first variable, which is…? Yes, that's the question. If the HP49G means that the first variable of $F(X(T))$ is $X(T)$ then the result is OK. But if it means that it is T, then the result is wrong! This is one of the problems that occur to us, uninformed users, if on the one hand, notions like $F(X(T))$ or $F\left(\frac{X}{Y}, \frac{Y}{X}\right)$ are allowed, but on the other hand the notion $dnF(X(T))$, or $dnF\left(\frac{X}{Y}, \frac{Y}{X}\right)$ is used to represent formal derivatives, where n gives the n^{th} variable of the function. If only functions of *names as variables* were allowed, like for example $F(X,Y)$, then we could always say what the first and what the second variables are. But if we allow such things like $F\left(\frac{X}{Y}, \frac{Y}{X}\right)$, go figure out what the first variable of that expression is. (If you tend to say that the first variable of $F\left(\frac{X}{Y}, \frac{Y}{X}\right)$ is X then wait until we examine such formal derivatives in much more detail.)

Especially for such expressions like $F(X(T))$, we can't use $X(T)$ as the variable of differentiation when we use ∂ because this function only accepts as arguments a function (or name) on stack level 2 and a *name* on stack level 1. This means also that it can't be used for finding such things, like for example

$$\frac{\partial}{\partial SIN(X)}(SIN(X))$$

without any further manipulations. This example could be solved by entering $\frac{\partial}{\partial T}(T)$, then entering the list $\{T\ \ SIN(X)\}$, and then using the function |. This returns the correct result 1. Note that entering $\frac{\partial}{\partial T}(T)$, then $T = SIN(X)$ and pressing SUBST will error out. We will examine such strange looking derivatives in more detail later on, and we will return to them when we take a look at derivatives of parametric functions.

We go a little further examining formal derivatives. They always appear when the HP49G doesn't know how to take the derivative of a function. This happens when some undefined abstract function has to be differentiated. If the definition of $F(X)$ doesn't exist in the current path, then using ∂ to take the derivative for X, will return $d1F(X)$. This means, as already said, the derivative of $F(X)$ for its first variable,

[1] ACO and the professor.

which is X. It is really amazing how much can be done with such expressions. Enter for example F(S), then Y, and press ∂ to find the derivative. The same considerations like on the previous pages, are the explanation for the result that we get. Since S contains SIN(Y), the result is COS(Y) d1F(SIN(Y)). The HP49G used the chain rule to return the product of the derivative for Y of the inner function SIN(Y), with the derivative of the outer function. As the latter is totally undefined, the HP49G returns d1F(SIN(Y)), to denote that derivative in a general abstract way. If on the other hand, we evaluate or expand $\frac{\partial}{\partial Y}(F(S))$, we get 0. In the first case, using RPL syntax, the function ∂ evaluated the variable S in F(S), and the result was F(COS(Y)). Then, the derivation for Y was carried out, and COS(Y) d1F(SIN(Y)) was found. In the second case, using algebraic syntax, The HP49G didn't care to evaluate S in F(S) first, and so the derivation for Y returned 0. At this point, it would be better to collect all the cases in a single table, for a better understanding of the behaviour of the function ∂. I made two tables for this. The first is on the next page and contains cases of differentiation of an expression or a user defined function. The second table is on the page after the next and contains the cases of differentiation of a function that isn't defined, that is cases of formal differentiation. In both tables, regions with the same colour are those which return their results using the same mechanism. That means, that it is not the same result that makes up a region, but rather the same *way* that is followed by the HP49G to return these results. The formal derivatives, will be also our entrance to yet another neglected feature of the HP49G, its further capabilities for handling formal derivatives in combination with user defined derivatives. He, he, this machine has much too much stuff that we forgot about. But what a marathon would it be without presenting exactly this stuff?

Before we take a look to those forgotten features, let's do some examples that are more complex. We use what we know until now, to explain how the results are derived. Keep all variables and user defined functions because the following examples use them.

Enter F(V(X)), then enter 'R' (in quotes) and press ∂. The result is 0, as we expect, because as we have seen already, the user defined function V(X) (argument of F) is evaluated before differentiation. This gives:

$$F\left(\frac{X^3}{X-1}\right)$$

Now, the HP49G sees that this function, though undefined, doesn't depend on R, because R doesn't appear as an argument of F. So the derivative is found to be 0.

If you enter F(V(X)) again, but then take the derivative with respect to X using the function ∂, then the result is:

$$\frac{(X-1)\ 3\ X^2 - X^3}{SQ(X-1)}\ d1F\left(\frac{X^3}{X-1}\right)$$

How is that result produced? Let's follow again our known rules. First of all the function V(X) is evaluated. The result of this action is:

$$\frac{X^3}{X-1}$$

This is used as argument for the undefined function F, that is we get:

$$F\left(\frac{X^3}{X-1}\right)$$

This result is then differentiated for X. According to the rules of

Expression \ Variable of derivation	RPL Syntax: Enter expression then variable, then use ∂ 'R'	RPL Syntax: Enter expression then variable, then use ∂ 'X'	Algebraic Syntax: Enter $\frac{\partial}{\partial \text{var}}(\text{Expression})$, then evaluate or expand 'R'	Algebraic Syntax: Enter $\frac{\partial}{\partial \text{var}}(\text{Expression})$, then evaluate or expand 'X'	
R^2+R (X^2 stored in R)	$2\ R+1$	0	$2\ R+1$	0	No evaluation of expression variables before differentiation.
V(X) (User function V defined as: << -> X 'X^3/(X-1)' >>)	0	$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}$	0	$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}$	Evaluation of function before differentiation.
V(QUOTE(X)) (User function V defined as: << -> X 'X^3/(X-1)' >>)	0	$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}$	0	* $\frac{2\ X^3-3\ X^2}{X^2-2\ X+1}$	Evaluation argument of function and of function before differentiation.
V(R) (User function V defined as: << -> X 'X^3/(X-1)' >> , X^2 stored in R)	0	$\frac{(X^2-1)\ 6\ X^5-X^6\ 2\ X}{SQ(X^2-1)}$	$\frac{(R-1)\ 3\ R^2-R^3}{SQ(R-1)}$	0	Evaluation of function but not of its argument before differentiation.
V(QUOTE(R)) (User function V defined as: << -> X 'X^3/(X-1)' >> , X^2 stored in R)	$\frac{(R-1)\ 3\ R^2-R^3}{SQ(R-1)}$	0	** $\frac{2\ R^3-3\ R^2}{R^2-2\ R+1}$	0	

* Result is $\frac{(QUOTE(X)-1)\ 3\ QUOTE(X)^2\ d1QUOTE(X)-QUOTE(X)^3\ d1QUOTE(X)}{SQ(QUOTE(X)-1)}$

** Result is $\frac{(QUOTE(R)-1)\ 3\ QUOTE(R)^2\ d1QUOTE(R)-QUOTE(R)^3\ d1QUOTE(R)}{SQ(QUOTE(R)-1)}$

EVALuate again to get the result in the table

derivation we get:

$$\frac{(X-1)\ 3\ X^2 - X^3}{SQ(X-1)} \quad d1F\left(\frac{X^3}{X-1}\right)$$

where the expression:

$$d1F\left(\frac{X^3}{X-1}\right)$$

stands for the formal derivative of:

$$F\left(\frac{X^3}{X-1}\right)$$

Variable of derivation / Expression	RPL Syntax: Enter expression then variable, then use ∂ — 'R'	RPL Syntax: 'X'	Algebraic Syntax: Enter $\frac{\partial}{\partial var}(Expression)$, then evaluate — 'R'	Algebraic Syntax: 'X'
$F(X)$	0	$d1F(X)$	0	$d1F(X)$
$F(X^2)$	0	$2\ X\ d1F(X^2)$	0	$2\ X\ d1F(X^2)$
$F(QUOTE(X))$	0	$d1F(X)$	0	* $d1F(QUOTE(X))$
$F(R)$ X^2 stored in R	0	$2\ X\ d1F(X^2)$	$d1F(R)$	0
$F(R^2)$ X^2 stored in R	0	$4\ X^3\ d1F(X^4)$	$2\ R\ d1F(R^2)$	0
$F(QUOTE(R))$ X^2 stored in R	$d1F(R)$	0	*** $d1F(QUOTE(R))$	0

Formal differentiation with expansion of intermediate differential forms.

Expansion of differential forms like $d1F(QUOTE(X))$ to $d1F(X)$.

No expansion of the differential forms like $d1F(QUOTE(X))$. Using EXPAND instead of two EVALs will expand such forms to forms like $d1F(X)$.

Evaluation of the argument of the undefined function before differentiation.

No evaluation of the argument of the undefined function before differentiation.

No evaluation of the argument of the undefined function before differentiation. No expansion of the differential forms like $d1F(QUOTE(R))$ Using EXPAND instead of two EVALs will expand such forms to forms like $d1F(X^2)$.

* EVALuate again to get $d1F(X)$

** EVALuate again to get $d1F(X^2)$

*** EVALuate again to get $d1F(R)$. An additional EVAL will return $d1F(X^2)$

That means a derivative which can't be explicitly found, since F is undefined.

We make the last example trying to find the derivative of $V(F(R))$ for R and for X. If you enter $V(F(R))$, then 'R', and then press ∂, you are going to get 0. This shows again, that the argument of F, which is R, gets evaluated to its contents, which are X^2. After this we have $V(F(X^2))$, which differentiated for R must return 0, since it doesn't depend on R. But if you enter $V(F(R))$, then X, and then press ∂, you get:

$$\frac{(F(X^2)-1)\ 3\ F(X^2)^2\ 2\ X\ d1F(X^2)-F(X^2)^3\ 2\ X\ d1F(X^2)}{SQ(F(X^2)-1)}$$

Let's see how this was produced. First, the argument of F was evaluated. This argument was R, and since X^2 is stored in R, the result of the evaluation was X^2. This was used as argument for F, and since F is undefined, we simply get $F(X^2)$. This expression was used as argument for V. So instead of getting

$$\frac{X^3}{X-1}$$

we got

$$\frac{F(X^2)^3}{F(X^2)-1}$$

Next, the differentiation for X was carried out. According to the rules of differentiation for a ratio, we have:

$$\frac{\partial}{\partial X}\left(\frac{F(X^2)^3}{F(X^2)-1}\right)=\frac{(F(X^2)-1)\ \frac{\partial}{\partial X}(F(X^2)^3)-F(X^2)^3\ \frac{\partial}{\partial X}(F(X^2)-1)}{SQ(F(X^2)-1)}$$

If we carry out the derivations of the right hand side of the last equation, keeping in mind that $d1F(X^2)$ is the formal derivative of $F(X^2)$, then we see that the result returned by the HP49G was correct (inside the frame of its own evaluation rules).

Do some examples for yourself and try to predict the behaviour of the function ∂, using the knowledge that we have so far. Remember that QUOTE can also be used with formal functions, which means that $F(QUOTE(R))$ is perfectly OK.

We proceed with some unexpected features regarding such formal derivations. First of all, the meaning the expressions $d1F(X)$ is: The derivative of F, for its first variable. "First" means really the order of appearance of the variable inside the parentheses. We make an example. Enter:

$$\frac{\partial}{\partial X}(F(Y,X))$$

and expand. This will return $d2F(Y,X)$, which means the derivative of F for its second variable, which is X. Such derivatives add really great power to the HP49G and they are not only of cosmetic nature. Suppose for example that you have the expression

$$\frac{\partial}{\partial X}(F(X)\ G(X))$$

Expanding this you will get the result $G(X)\ d1F(X)+F(X)\ d1G(X)$. This result has been calculated using the product rule of derivation. If you have the derivative of some complex expression, in which many

functions are combined in many different ways, then this feature can break the derivative into many small pieces, each of which contains a derivative of a single function for a single variable. Consider for example:

$$\frac{\partial}{\partial X}\left(\frac{F(X)\ G(X)^2}{G(X)+X}\right)$$

In this form it is really hard to say how the derivatives of the functions participate to built-up the whole derivative. Bur expanding this, you get the result:

$$\frac{\left(G(X)^3 - X\ G(X)^2\right)\ d1F(X) + \left(G(X)^2 + 2\ X\ G(X)\right)\ F(X)\ d1G(X) - G(X)^2\ F(X)}{G(X)^2 + 2\ X\ G(X) + X^2}$$

This contains only derivatives of a single function for a single variable, which makes easier to see how the differential forms participate to built-up the derivative

$$\frac{\partial}{\partial X}\left(\frac{F(X)\ G(X)^2}{G(X)+X}\right)$$

If you have taken some thermodynamics class, then you surely know how easier life can get with this feature. We will have some examples on this later on, when we know enough about derivatives on the HP49G.

Another very special feature of such formal derivatives is that in some sense they are not special at all! Enter for example `d1F` and press TYPE to find out that this is simply a name with object type 6. What does this imply? Well, sometimes we know the derivative of some function but we don't need the function itself for our work. On the HP49G we can not only define functions but also derivatives, that is, we can make not only user defined functions but also *user defined derivatives*. Suppose for example that we know that the derivative of $F(X,Y)$ for X is $e^{X^2\ Y}\ (X - Y - 2)$. We go to the EQW, we enter $d1F(X,Y) = e^{X^2\ Y}\ (X - Y - 2)$, we press ENTER to put the equation on the stack and we press DEF. Then a new variable `d1F` is created, which contains:

```
<< → X Y 'EXP(X^2*Y)*(X-Y-2)' >>
```

Later on, we might have to calculate something like for example:

$$\frac{\left(\frac{\partial}{\partial X}\left(F(X,Y)\right)\right)^2}{e^{X\ Y}}$$

If you expand this without having made the definition of $d1F(X,Y)$, you will get:

$$\frac{d1F(X,Y)^2}{e^{Y\ X}}$$

But if the derivative $d1F(X,Y)$ is defined in the current path, you get:

$$\frac{\left(e^{X^2\ Y}\ (X - Y - 2)\right)^2}{e^{Y\ X}}$$

If you now press LIN and then FACTOR, you get:

$$e^{2\ Y\ X^2 - Y\ X}\ \left(X - (Y + 2)\right)^2$$

The user defined derivative $d1F(X,Y)$ was evaluated just like any other normal function. It used the arguments X and Y and returned the result $e^{X^2\ Y}\ (X - Y - 2)$ according to its definition.

Alternatively you can use it also in RPL syntax. Enter the arguments X and Y and press **d1F** to get the result $e^{X^2\ Y}\ (X-Y-2)$.

We had good news until now, so it's time for some bad news. There are also problems with the notation like $d1F(X)$. One of the problems is, as we already noticed, that we are allowed to build-up things like $d1G\left(\begin{matrix}X\\Y\end{matrix}\right)$ that don't have a distinct order of their arguments. At the same time the information about the variable of differentiation is coded as a number between the small "d" and the name of the function. In the above example, which is the first variable? If you say that it is X, then let's have an example to convince you about the problem. Enter

$$\frac{\partial}{\partial X}\left(e^{G\left(\begin{matrix}X\\Y\end{matrix}\right)}\right)$$

Press **EXPAND** to get:

$$\frac{d1G\left(\begin{matrix}X\\Y\end{matrix}\right)}{Y}$$

This agrees with the theory that the first variable is X, because we differentiated for X and we got an answer that contains:

$$d1G\left(\begin{matrix}X\\Y\end{matrix}\right)$$

But according to this, if we differentiate for Y, we must get an answer that contains:

$$d2G\left(\begin{matrix}X\\Y\end{matrix}\right)$$

Enter:

$$\frac{\partial}{\partial Y}\left(e^{G\left(\begin{matrix}X\\Y\end{matrix}\right)}\right)$$

and expand. You get:

$$-\frac{X\ d1G\left(\begin{matrix}X\\Y\end{matrix}\right)}{Y^2}$$ (Gasp!!!)

If X is the first variable, why then is Y ... also the first variable? The result contains:

$$d1G\left(\begin{matrix}X\\Y\end{matrix}\right)$$

which is the same like before, the derivative of $G\left(\begin{matrix}X\\Y\end{matrix}\right)$ for its first variable, though we differentiated for Y and not for X this time.

The notation:

$$\frac{\partial}{\partial X}(\text{Function}(\text{arguments}))$$

is much more precise than the notation $d1\text{Function}(\text{arguments})$ because the first explicitly shows the variable with respect to which we are taking the derivative, no matter if it is the first, the second or the twentieth.

This shows that the notation $d1\text{Function}(\text{arguments})$ is only thought for a listing of arguments, be them simple names or expressions, inside the parentheses. For example evaluating

$$\frac{\partial}{\partial Y}\left(G(X,Y)\right)$$

or

$$\frac{\partial}{\partial X}\left(G(X,Y)\right)$$

will return results that can be interpreted easily, like $d2G(X,Y)$ or $d1G(X,Y)$. When some argument is not a simple name, like for example in

$$\frac{\partial}{\partial X}\left(G\left(\frac{X}{Y}\right)\right)$$

we run into troubles, because of the ambiguous interpretation of the expression:

$$d1G\left(\frac{X}{Y}\right)$$

in the result.

From the above I might have created the impression that for example the expressions

$$\frac{\partial}{\partial X}\left(G\left(X^2\right)\right)$$

and $d1G\left(X^2\right)$ are at least theoretically equal. But this isn't true (at least in the CAS world of the HP49G). If you enter

$$\frac{\partial}{\partial X}\left(G\left(X^2\right)\right)$$

and expand, you get $2\ X\ d1G\left(X^2\right)$. This means that for the HP49G the relation holds:

$$\frac{\partial}{\partial X}\left(G\left(X^2\right)\right) = 2\ X\ d1G\left(X^2\right)$$

The two expressions,

$$\frac{\partial}{\partial X}\left(G\left(X^2\right)\right)$$

and

$$2\ X\ d1G\left(X^2\right)$$

differ by a factor of $2\ X$, and so the expressions

$$\frac{\partial}{\partial X}\left(G\left(X^2\right)\right)$$

and

$$d1G\left(X^2\right)$$

are not identical! Only in cases where the function has simple names as arguments, like for example $G(X,Y)$, the two notations

$$\frac{\partial}{\partial X}\left(G(X,Y)\right)$$

and

$d1G(X,Y)$ are equivalent. The expression:

$$\frac{\partial}{\partial X}\left(G\left(X^2\right)\right)$$

means the derivative of $G\left(X^2\right)$ according to the chain rule, that is the product of the "inner" derivative with the "outer" derivative. The expression $d1G\left(X^2\right)$ means the derivative of $G\left(X^2\right)$ with respect to… X^2(!!!) Now we can perhaps see better, what is meant by first, second, and so on arguments of such expressions. The expression:

$$dnG\left(arg_1, arg_2, \quad , arg_m\right)$$

means the derivative of G for its n^{th} argument, counting the "slots" between the commas and not the names of variables that appear in the arguments, which themselves can be arbitrary expressions. That means, that for example:

$$d2G\left(SIN(X), COS(X)\right)$$

is equivalent to the expression:

$$\frac{\partial}{\partial COS(X)}\left(G\left(SIN(X), COS(X)\right)\right)$$

which is impossible to write *directly* with the HP49G! It it also totally impossible to do? Well, let's see. First, we have to get an idea of what such a derivative means.

A derivative $\frac{\partial f(x)}{\partial x}$ of a function $f(x)$ can be understood as the rate of change of $f(x)$ in relation to the rate of change of x. This means that knowing it, we also know how the values of $f(x)$ alter when x itself alters. The notion $f(x)$ shows that the function f depends on x.

When x changes, then $f(x)$ (in general) also changes. But we can also consider how some function $f(x)$ changes when some other function $g(x)$ changes. For example, we can consider what $SIN(X)$ does, with corresponding changes of $COS(X)$. And this can be written as:

$$\frac{\partial}{\partial COS(X)}\left(SIN(X)\right)$$

How to deal with such expressions?

Since both things depend on X, we can use parametrisation. We define $t = COS(X)$, where t is a new variable, defined as a function of the variable X. Enter the equation $t = COS(X)$. Now, if we solve $t = COS(X)$ for X, we are going to get X as a function of t. Enter X and press SOLVE. The result is:

$$\left\{X = -\left(2\ n1\ \pi + ACOS(t)\right)\quad X = 2\ n1\ \pi + ACOS(t)\right\}.$$

This can be substituted in $SIN(X)$, in order to convert it to a function of t. Enter $SIN(X)$, press ▶ and then SUBST. Now you have:

$$\left\{SIN\left(-\left(2\ n1\ \pi + ACOS(t)\right)\right)\quad SIN\left(2\ n1\ \pi + ACOS(t)\right)\right\}.$$

Now, if we consider the expression

$$\frac{\partial}{\partial COS(X)}\left(SIN(X)\right)$$

again, we see that $SIN(X)$ can be replaced by either:

$$SIN\left(-\left(2\ n1\ \pi + ACOS(t)\right)\right)$$

or:

$$\mathrm{SIN}(2\ n1\ \pi + \mathrm{ACOS}(t))$$

And $\mathrm{COS}(X)$ can be replaced by t. So the derivative

$$\frac{\partial}{\partial \mathrm{COS}(X)}(\mathrm{SIN}(X))$$

can be written as:

$$\frac{\partial}{\partial t}(\mathrm{SIN}(-(2\ n1\ \pi + \mathrm{ACOS}(t))))$$

or:

$$\frac{\partial}{\partial t}(\mathrm{SIN}(2\ n1\ \pi + \mathrm{ACOS}(t))).$$

With the list:

$$\{\mathrm{SIN}(-(2\ n1\ \pi + \mathrm{ACOS}(t)))\ \ \mathrm{SIN}(2\ n1\ \pi + \mathrm{ACOS}(t))\}$$

on stack level 1, enter t and press ∂ to get the list:

$$\begin{matrix} \mathrm{COS}((2\ n1\ \pi + \mathrm{ACOS}(t)))\ \ \dfrac{1}{\sqrt{1-\mathrm{SQ}(t)}} \\ \mathrm{COS}(2\ n1\ \pi + \mathrm{ACOS}(t))\ \ -\dfrac{1}{\sqrt{1-\mathrm{SQ}(t)}} \end{matrix}$$

Press OBJ→ to explode the list. Press ◀ to get rid of the element count. Now the expression

$$\mathrm{COS}(2\ n1\ \pi + \mathrm{ACOS}(t))\ \ -\frac{1}{\sqrt{1-\mathrm{SQ}(t)}}$$

is on stack level 1. Press TEXPAND to get :

$$-\left(t\ \mathrm{COS}(2\ n1\ \pi)+\sqrt{-(t^2-1)}\ \mathrm{SIN}(2\ n1\ \pi)\right)\frac{1}{\sqrt{1-\mathrm{SQ}(t)}}$$

Since the HP49G still has no integer assuming capabilities, we must do a bit of work by hand. Press ▼ to get this in the EQW. Since $n1$ is integer we know that $\mathrm{COS}(2\ n1\ \pi)$ is equal to 1 and $\mathrm{SIN}(2\ n1\ \pi)$ is equal to 0. Edit the expression and change it to:

$$-\left(t\ 1-\sqrt{-(t^2-1)}\ 0\right)\frac{1}{\sqrt{1-\mathrm{SQ}(t)}}$$

Press ENTER to put it on the stack. Expand it to get:

$$\frac{t\ \sqrt{-(t^2-1)}}{t^2-1}$$

Press ▶ and follow the same instructions to change the expression:

$$\mathrm{COS}((2\ n1\ \pi + \mathrm{ACOS}(t)))\ \ \frac{1}{\sqrt{1-\mathrm{SQ}(t)}}$$

to:

$$-\frac{t\ \sqrt{-(t^2-1)}}{t^2-1}$$

Now we rebuild the list. Enter 2 and press →LIST to get:

$$\begin{bmatrix} t\sqrt{-(t^2-1)} & -t\sqrt{-(t^2-1)} \\ t^2-1 & t^2-1 \end{bmatrix}$$

These are the two possible results of

$$\frac{\partial}{\partial \mathrm{COS}(X)}(\mathrm{SIN}(X))$$

written as functions of the variable t, for which we have defined $t = \mathrm{COS}(X)$. We can use this formula to do back substitution. Enter $t = \mathrm{COS}(X)$ and press SUBST to get:

$$\begin{bmatrix} \mathrm{COS}(X)\sqrt{-(\mathrm{COS}(X)^2-1)} & -\mathrm{COS}(X)\sqrt{-(\mathrm{COS}(X)^2-1)} \\ \mathrm{COS}(X)^2-1 & \mathrm{COS}(X)^2-1 \end{bmatrix}$$

Press TRIGSIN to convert $-(\mathrm{COS}(X)^2-1)$ to $\mathrm{SIN}(X)^2$ and get:

$$\begin{bmatrix} -\dfrac{\mathrm{COS}(X)\,|\mathrm{SIN}(X)|}{\mathrm{SIN}(X)^2} & \dfrac{\mathrm{COS}(X)\,|\mathrm{SIN}(X)|}{\mathrm{SIN}(X)^2} \end{bmatrix}$$

The two expressions in the list are the results of the differentiation:

$$\frac{\partial}{\partial \mathrm{COS}(X)}(\mathrm{SIN}(X))$$

Leave them on the stack, as we are going to use them in some minutes.

If we want to visualise the above, then we can plot all points that have as x-coordinate $\mathrm{COS}(X)$ and as y-coordinate $\mathrm{SIN}(X)$. It sounds familiar, doesn't it? Yes, this is the built-in plot type Parametric. Perhaps now it is more clear, that we used *parametrisation* to find the above derivative. Let's see how we do parametric plots on the HP49G. First of all, for two dimensional parametric plots the HP49G uses complex quantities. The real part is used for plotting the horizontal coordinate, and the imaginary part is used for plotting the vertical coordinate. For example, if you have some parametric function, like:

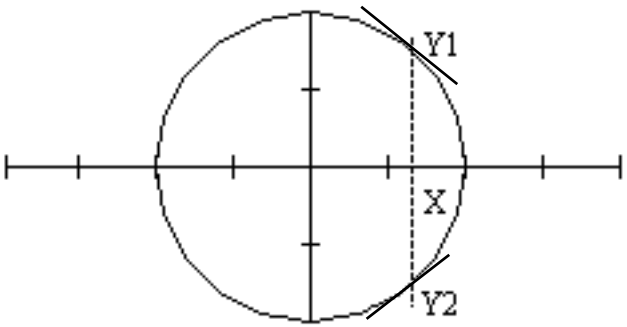
$$Y(X)\begin{cases} X = t^2 \\ Y = t-1 \end{cases}$$

where the coordinates X and Y depend on the parameter t, then the complex quantity that the HP49G plots, is $t^2+(t-1)\,i$. That's what you have to store in EQ, in order to use the plot type Parametric. In our example we have found how $y = \mathrm{SIN}(X)$ changes, when $x = \mathrm{COS}(X)$ changes. The horizontal and vertical coordinates depend on the parameter X. So we will plot $\mathrm{COS}(X)+\mathrm{SIN}(X)\,i$.

Press and hold down ◄ and while you hold this key down, press F4. The PLOT SETUP screen appears. Choose Parametric plot type. Move to the input field EQ: and enter $\mathrm{COS}(X)+\mathrm{SIN}(X)\,i$. Enter X in the input field Indep:, since X is our parameter. Note that in this type of plot, Indep: is *not the horizontal coordinate!* Now, press and hold down ◄ and while you hold this key down, press F2 to go to the PLOT WINDOW − PARAMETRIC screen. Set horizontal view from -2 to 2, and vertical view from -1 to 1. Set Low: to 0 and High: to 6.28. (This is approximately from 0 to $2\,\pi$.) Set Step: to 0.314. (Approximately $\frac{\pi}{10}$.) Now press ERASE and then DRAW. You get a circle, that starts at $(\mathrm{COS}(0),\mathrm{SIN}(0)) = (1,0)$, goes once

around the plot origin in clockwise direction, and ends again at $(1,0)$. The slope of the curve at any point of the circle is given by what we have found, that is by both formulae in the list:

$$\begin{matrix}\{ & -\dfrac{\mathrm{COS}(X)\ |\mathrm{SIN}(X)|}{\mathrm{SIN}(X)^2} & \dfrac{\mathrm{COS}(X)\ |\mathrm{SIN}(X)|}{\mathrm{SIN}(X)^2} & \}\end{matrix}$$



In this list, the variable X is no more a parameter but a coordinate, namely the horizontal coordinate. This is exactly why we have *two* formulae for the slope. If you take an arbitrary value for the X coordinate between -1 and 1, then the are two vertical coordinates Y that correspond to it. And so we also have two derivatives. We want to superimpose the plot of the derivatives with the parametric plot. The derivatives are *functions* and so we might be inclined to change the plot type to Function. But this would cause problems. While the horizontal coordinate of the parametric plot goes from -1 to 1, the corresponding horizontal coordinate of the functions should be from 0 to 6.28, *exactly like the values for the parameter* X *of the parametric plot. This* would plot the derivatives in a way that corresponding horizontal coordinates of the parametric plot and its derivatives wouldn't coincide. We must find another way to plot the derivatives. One option is to plot them also as parametric functions. Consider for example the first formula in the above list, which defines the function:

$$y = -\frac{\mathrm{COS}(X)\ |\mathrm{SIN}(X)|}{\mathrm{SIN}(X)^2}$$

We can turn it to a parametric function by writing:

$$x = \mathrm{COS}(X)$$

$$y = -\frac{\mathrm{COS}(X)\ |\mathrm{SIN}(X)|}{\mathrm{SIN}(X)^2}$$

Note that we didn't simply set $x = X$, because we want to stay compatible to the first plot, in which the horizontal coordinate x was set equal to $\mathrm{COS}(X)$. (The variables x and X are not the same!) Our parametric representation of the first derivative is:

$$\mathrm{COS}(X) + \left(-\frac{\mathrm{COS}(X)\ |\mathrm{SIN}(X)|}{\mathrm{SIN}(X)^2}\right) i$$

Since we have two derivatives, we must also plot two parametric functions. That means that we must enter the list:

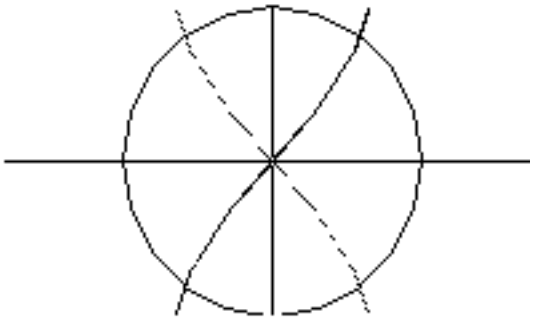
$$\begin{matrix}\{ & \mathrm{COS}(X) + \left(-\dfrac{\mathrm{COS}(X)\ |\mathrm{SIN}(X)|}{\mathrm{SIN}(X)^2}\right) i & \mathrm{COS}(X) + \dfrac{\mathrm{COS}(X)\ |\mathrm{SIN}(X)|}{\mathrm{SIN}(X)^2}\ i & \}\end{matrix}$$

in the input field EQ: of the PLOT SETUP screen. (When the reserved variable EQ contains a list of parametric functions rather than a single parametric function, then all parametric functions are plotted together.) Press CANCEL to leave the plot, and then ← F4 to go back to the PLOT SETUP screen. Select the input field EQ:. Because entering a function can be a tedious task without the EQW, and because you can't enter the list directly in the EQW, you can temporarily leave the PLOT SETUP screen, build-up the list using the stack, and return to the PLOT SETUP screen with the list later. Press HIST to go to

the interactive stack, which still contains the list with the two derivatives. The small arrow at the right of the stack level number 1 indicates that the object in this stack level can be used for further operations. Press ECHO. This echoes the list to the command line, which we came from. (The command line of the input field EQ:.) Press CANCEL to leave the interactive stack and return to the PLOT SETUP screen. Now you see that the list waits in the command line to be entered in the input field "EQ:". Press [ENTER] to put it in the input field EQ:. This is not exactly the list that we have to plot, so we must edit it. Select the menu input field EQ: and press NXT until you see CALC over the key F2. Press CALC to temporarily leave the PLOT SETUP screen. Now you are on the stack. This is not the same operation like pressing HIST, as it copies the contents of the selected input field on a *new empty stack*. It also starts the *normal* stack environment instead of sending you to the *interactive* stack. This new stack can be thought a separate private stack of the input field EQ:. The global stack is also preserved but is temporarily hidden, so don't worry as we are not going to lose anything. The top part of the screen still shows the title of the input screen that you came from, as an aid for letting you know exactly where you are. Also, the message Enter function(s) to plot appears on the top of the screen to indicate exactly which input field will receive what you enter now. The previous contents of the input field EQ: are put on the stack. We have to change this list to:

$$\left\{ \cos(x) + \left(-\frac{\cos(x)\,|\sin(x)|}{\sin(x)^2}\right) i \quad \cos(x) + \frac{\cos(x)\,|\sin(x)|}{\sin(x)^2} \right\}$$

We are going to use the list processing capabilities of the HP49G. Enter i and press × to multiply both objects in the list with the imaginary unit. Now, enter COS(X) and press ▶ to swap stack levels 1 and 2. You might think now that you just have to press + to add COS(X) to both objects in the list. But especially this operation concatenates two lists or any object with a list, rather than adding an object to all objects in a list. For this purpose there is the command ADD. It is the sixth item of the menu MTH/LIST, so press MTH, and then press LIST. Now press ADD to add COS(X) to both algebraic objects in the list. The message on the top of the screen has changed to Press [CONT] for menu. Press CONT to return to the menu that will allow to take the list on stack level 1 to the input field EQ:. Now press OK to return to the PLOT SETUP screen and put the list to that input field. Press DRAW. The two derivatives are plotted on the same plot that contained the circle. Press CANCEL to return to the (global) stack. As you can see, it still contains the original (not the edited) list with the derivatives.



In order to find such derivatives, like for example

$$\frac{\partial}{\partial \cos(x)}(\sin(x))$$

we don't need to do all by hand, like we did on pages 1-24 to 1-26. We can make a program that does (almost) everything automatically. Consider the code:

```
<< → f1 f2                @Store in locals
  << f1 f2                @Recall locals
    IF f2 TYPE 6. ==      @If f2 is a name
    THEN                  @then
      ∂                   @take derivative
    ELSE                  @else
      LNAME 1 GET         @get first name that is in f2,
      SWAP ttemp =        @build up equation f2=ttemp
      SWAP SOLVE          @solve equation for first name
      SUBST EXPAND        @substitute in f1
      ttemp ∂             @take derivative for ttemp
```

```
      ttemp f2 = SUBST  @back substitution
      EXPAND            @expand
    END
  >>
>>
```

Store the program in dF1F2 and let's test it. The program takes two arguments. On stack level 2 it expects to find the expression to differentiate. In stack level 2 it expects the expression that specifies what to differentiate for.

Enter X^2 and make a copy of this expression on stack level 2. Press the menu key dF1F2. The result is the list $\{1\ \ 1\}$, which shows that it worked. (Anything differentiated for itself has to return 1.) Perhaps code should be added to remove duplicates from the list?

Enter e^{X^2} and then e^X. Press dF1F2 again to get $\frac{2\ X\ e^{X^2}}{e^X}$.

And one example in algebraic syntax. Go to the EQW and enter dF1F2$\left(e^{X^2},\frac{1}{X}\right)$. Put that on the stack and EVALuate to get $-\left(2\ \ X^3\right)$.

Let's try also the example that we used for parametric plots. Enter SIN(X) and then COS(X). Press dF1F2. After some seconds the HP49G returns the list:

$$\left\{ -\frac{\sqrt{-\left(\mathrm{COS}(X)^2-1\right)}\ \ \mathrm{COS}\left(\mathrm{ACOS}\left(\mathrm{COS}(X)+2\ \ n1\ \pi\right)\right)}{\mathrm{COS}(X)^2-1}\ \ -\frac{\sqrt{-\left(\mathrm{COS}(X)^2-1\right)}\ \ \mathrm{COS}\left(\mathrm{ACOS}\left(\mathrm{COS}(X)+2\ \ n1\ \ \pi\right)\right)}{\mathrm{COS}(X)^2-1}\right\}$$

As you can see, the program didn't do the work that we did in the example. But this would be too much to demand as there is a huge number of possible results and possibilities to simplify them. The real problem is that (as already said 100 times) we still have no integer assumptions on the HP49G. In this case you can enter n1= 1 and then press SUBST, to turn n1 from a name to an integer. Then press SIMPLIFY to get:

$$\left\{ \frac{\mathrm{COS}(X)\ \ |\mathrm{SIN}(X)|}{\mathrm{SIN}(X)^2}\ \ -\frac{\mathrm{COS}(X)\ \ |\mathrm{SIN}(X)|}{\mathrm{SIN}(X)^2}\right\}$$

The program uses LNAME to find the first name returned in the vector of names that appear in f2. It assumes that this is the variable whose variation causes the variations of f1 and f2. But this doesn't have to be always true. Instead of using the command LNAME, we could also give an additional argument to the program, to specify the varying variable. Another problem is that the program relies on SOLVE to solve the equation f2 = ttemp. If this step fails, then the program will not work. Since we will talk in much more detail about derivatives of functions in parametric form, we leave the program dF1F2 as it is for the time being.

A question that has been asked quite often in the group, is how to do implicit differentiation on the HP49G. And the answer to this question is rather simple: Use exactly the same method like for explicit differentiation. Suppose for example that you have the implicit function given by SIN(Y) = Y − X. In this formula the variable Y means actually Y(X), the quantity Y is a function of the quantity X. We can't solve the equation SIN(Y) = Y − X analytically for Y. That means, we can't write Y(X) = someFunction(X). Nonetheless we can find the derivative of Y(X) as a function of Y(X) and X by differentiating implicitly. To do that on the HP49G, we must always keep in mind that the calculator doesn't know that Y somehow depends on X. And in this case it is a very good policy to not know. Consider for example what

would happen if you had to take the derivative of X Y Z for X. If the HP49G would automatically consider all variables different than X to be functions of X, then we would end up with very complicated expressions containing all thinkable derivatives of all variables other than X. And we would need a special mechanism for denoting that some variables don't depend on X. Instead of this the HP49G considers all variables in an expression that don't explicitly depend on X (or any other variable of differentiation) as constants. If we have some variable for which we know that it does depend on X (or any other variable of differentiation), we have to write this explicitly. In the example $\text{SIN}(Y) = Y - X$, where Y is a variable that depends on X, we write $\text{SIN}(Y(X)) = Y(X) - X$. If you enter this, then enter X, and then press ∂, then you get $\text{COS}(Y(X))\ \text{d1Y}(X) = \text{d1Y}(X) - 1$. Now, you can enter $\text{d1Y}(X)$ and then press SOLVE, to get the solution:

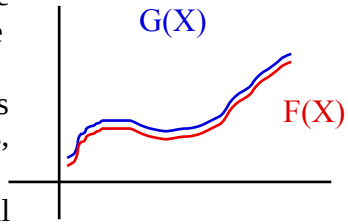
$$\text{d1Y}(X) = \frac{1}{\text{COS}(Y(X)) + 1}$$

The HP49G has done implicit derivation and solved for $\text{d1Y}(X)$. The last operation, solving for $\text{d1Y}(X)$ might look unfamiliar, since we solved not for a variable but for an algebraic expression, but it is exactly as good as solving for any variable. To get an idea of this capability, enter $\text{SIN}(X) + \text{COS}(Y(X)) = e^{X+Y(X)}$, then $\text{COS}(Y(X))$, and the press SOLVE to solve for $\text{COS}(Y(X))$. The result is $\text{COS}(Y(X)) = e^{X+Y(X)} - \text{SIN}(X)$, which shows that the whole expression $\text{COS}(Y(X))$ was considered as a single variable. Still about implicit derivation: If you enter the above example as $\text{SIN}(Y) = Y - X$, and take the derivative for X, then you are going to find $0 = -1$. This unusual result comes because the left and the right hand sides of the equation $\text{SIN}(Y) = Y - X$ were differentiated separately for X. Since the left hand side doesn't explicitly depend on X, the differentiation for X (i.e. $\frac{\partial}{\partial X}(\text{SIN}(Y))$) returned 0. On the other hand, the right hand side does depend on X, and so the differentiation for X (i.e. $\frac{\partial}{\partial X}(Y - X)$) returned −1. Both results were then set equal and so the result was $0 = -1$. This isn't a bug even if the resulting equation is impossible. This "impossibility" was already contained in the expression $\text{SIN}(Y) = Y - X$ in combination with derivation for X. The concept used in the HP49G for derivation, says that anything that doesn't explicitly depend on the variable of differentiation, is a constant. If Y is a constant in the equation $\text{SIN}(Y) = Y - X$, then the equation is not an identity but a proposition. It isn't valid for any value of Y but only for some particular values. In this case we can't conclude equality of derivatives because of general equality of the two sides of the equation, i.e., we can't say:

$$\text{leftHandSide}(Y,X) = \text{rightHandSide}(Y,X)$$

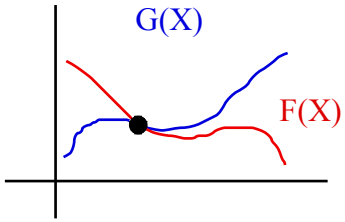
$$\frac{\partial}{\partial X}(\text{leftHandSide}(Y,X)) = \frac{\partial}{\partial X}(\text{rightHandSide}(Y,X))$$

To understand this better, consider the equation $F(X) = G(X)$. (In this example we only have the variable X, but for the understanding this doesn't matter.) If this equation is an identity, then the two things, $F(X)$ and $G(X)$ are *always* equal. If we plot them we are going to get two identical graphs. And since the two curves are identical, so must also be their derivatives. That means, in this case we can take the derivatives of both sides and set them equal. (Technique used in implicit differentiation.)



On the other hand, if the equation $F(X) = G(X)$ is not an identity but a proposition, then it will hold only for some particular values of X. Plotting the two things, $F(X)$ and $G(X)$, will produce two different

graphs, which (eventually) intersect at some particular values of X. We see that in this case the derivatives of the two things can't be equal (in general).



Thus, on the HP49G, we have the possibility to do both. If the equation which we use for implicit differentiation is an identity, we have to denote this by explicitly writing all dependencies on the variable of differentiation. If on the other hand, we have some equation that is a proposition, and we (for some reason) have to take the derivatives of the right and the left hand side and set them equal to each other, we omit the explicit dependencies on the variable of differentiation.

Last thing we are going to examine is how substitutions behave, when use on expressions that contain ∂. Enter:

$$\frac{\partial}{\partial X}\left(X^2-1\right)$$

then X = Y and then press SUBST. The result is 2 Y, which is correct, but doesn't allow us to tell if the substitution was made before or after the differentiation. Undo the last operation, and edit the equation in stack level 1 to $X = Y^2$. Press again SUBST. This time the HP49G errors out: Bad Argument Type. This shows that the HP49G tried to do the substitution *before* taking the derivative. It tried to substitute $X = Y^2$ for each occurrence of X in $\frac{\partial}{\partial X}\left(X^2-1\right)$. This would return:

$$\frac{\partial}{\partial Y^2}\left(\left(Y^2\right)^2-1\right)$$

and as we already have seen, this is impossible on the HP49G right out of the box. (That's why we made the program dF1F2.)

On the other hand, we have also the command | for substitutions. Drop the equation $X = Y^2$, and enter the list {X Y}. Press ⊓. The result is again the correct expression 2 Y, but now the operation took a bit longer, which shows that some other mechanism was used. Undo the operation, and edit the list to $\{X\ Y^2\}$. Press ⊓ again to get $2\ Y^2$. Wow, it worked! The HP49G has found the result of:

$$\frac{\partial}{\partial Y^2}\left(\left(Y^2\right)^2-1\right)$$

Does this make our program dF1F2 totally unnecessary? Unfortunately not, because for derivatives like for example

$$\frac{\partial}{\partial \mathrm{SIN}(X)}\left(\mathrm{COS}(X)\right)$$

we can't simply enter some $\frac{\partial}{\partial}(\ )$ and use some substitution list with |, to directly get the correct result. We have first to find some kind of parametrisation, which is what dF1F2 mainly does. In the case of

$$\frac{\partial}{\partial \mathrm{SIN}(X)}\left(\mathrm{COS}(X)\right)$$

as we already have seen, we can write:

$$\frac{\partial}{\partial t}\left(\sqrt{1-t^2}\right)$$

then enter {t SIN(X)}, and then press ⊓, to get the result:

$$\frac{-(2\ \mathrm{SIN}(X))}{2\ \sqrt{1-\mathrm{SIN}(X)^2}}$$

If you expand and then press TRIGCOS, you find:

$$-\frac{\mathrm{SIN}(X)\ |\mathrm{COS}(X)|}{\mathrm{COS}(X)^2}$$

which is exactly one fo the two results that dF1F2 returns, if we give it the arguments $\mathrm{COS}(X)$ and $\mathrm{SIN}(X)$. However, the built-in function | can be used directly for derivatives with respect to expressions $f(x)$, when the derivatives are of the form:

$$\frac{\partial}{\partial f(x)}\left(G(f(x))\right)$$

where $G(f(x))$ is a function that depends explicitly only on $f(x)$. Such derivatives are for example:

$$\frac{\partial}{\partial \mathrm{SIN}(X)}\left(\mathrm{SIN}(X)^2-\mathrm{SIN}(X)\right)$$

or:

$$\frac{\partial}{\partial X^2}\left(\mathrm{SIN}(X^2)-X^2\right)$$

and so on. We can't enter them directly since the quantity for which we differentiate isn't a single variable name. But we can enter first the derivative in which all occurrences of $f(x)$ are replaced by a single variable name, say S. Then we can enter the list $\{S\ \ f(x)\}$ and use the command | to find the derivative. For example, if we have

$$\frac{\partial}{\partial \mathrm{SIN}(X)}\left(\mathrm{SIN}(X)^2-\mathrm{SIN}(X)\right)$$

we can enter:

$$\frac{\partial}{\partial S}\left(S^2-S\right)$$

then the list $\{S\ \ \mathrm{SIN}(X)\}$, and then press |, to find the derivative $2\ \mathrm{SIN}(X)-1$.

Let's move on now to the other command that the HP49G provides for derivation, DERIV. This is the new command that came with the CAS, that means the HP48 calculators do not have it. (Except of course if you install ERABLE.) How does this command behave? You guessed right, we are going to repeat what we did with ∂, but now using DERIV. Take a breath and here we go.

In RPL syntax the command DERIV takes the function from stack level 2, and the variable from stack level 1, and returns the derivative of the function for the specified variable. For example, enter $e^{\mathrm{SIN}(Y^2)}$, then enter Y and then press DERIV. (The command is the second item in menu CALC/DERIV). The result is $e^{\mathrm{SIN}(Y^2)}\ \mathrm{COS}(Y^2)\ 2\ Y$. The same can be done using an algebraic syntax. Go to the EQW and press DERIV to write the unfinished expression DERIV(◀ ,■). The cursor blinks at the left of the comma to indicate that the HP49G expects you to enter the function that must be differentiated. Enter $e^{\mathrm{SIN}(Y^2)}$. Press ▶ to go to the second place holder to the right of the comma and enter Y. Now you have DERIV(æ$e^{\mathrm{SIN}(Y^2)}$,Yö. Press ENTER to put that on stack level

1. Let's see how the expression looks like when it isn't shown in pretty print. Press ◀ and then ▼ to edit the expression not in the EQW but in the command line, where no pretty print is used. Now you see: 'DERIV(EXP(SIN(Y^2)),Y)', which is quite different than the syntax of ∂. The general syntax of DERIV in the command line is 'DERIV(Function,Var)', where Var stands for the variable of derivation and Function for the function whose derivative you want to find. Press now ENTER to put the expression back to the stack. If you expand it, then you get the same result as before, $e^{SIN(Y^2)}\ COS(Y^2)\ 2\ Y$.

Can we find the slope of a function at a given point by finding the derivative and substitute the value of the variable at that point? Let's see. We try to find the slope of SIN(X) at X = π. In the EQW we type DERIV(SIN(X),X), and enter that on the stack. Then we enter X = π and press SUBST. The result is DERIV(SIN(π),π), which is not what we want, because the substitution has been carried out before the derivation took place. Obviously we can't use SUBST with DERIV this way. First we must explicitly expand the expression DERIV(SIN(X),X) to get COS(X), and then we can use SUBST for finding the slope. Let's see what happens when we use SUBST with DERIV in algebraic syntax. Go to the EQW and type SUBST(DERIV(SIN(X),X),X = π). Press ENTER to put his on the stack and press EXPAND. The result is COS(π) which is correct but not completely expanded. Obviously the expansion of the algebraic object SUBST(DERIV(SIN(X),X),X = π) retains the order of operations starting at the innermost sub expressions. Since the innermost operation is the derivation, it is carried out before substitution. But substituting in stack syntax just "puts the values" in the object of stack level 2, without first expanding it. Note also that in RPL mode you can't enter 'SUBST(DERIV(SIN(X),X),X = π)' from the command line. The built-in syntax checker will complain about a syntax error at the position of the "=". Strange? Well, it is even stranger that the erroneous 'SUBST(expression,val)' does not cause a syntax error, if val is for example some number or name, but not an equation! If you for some reason have to build up the algebraic object 'SUBST(DERIV(f(var),var),var = value)' in a program, then you have to do that in some other way. The code snippet below takes an expression and a substitution equation and returns 'SUBST(expression,substitutionEquation)'.

```
'SUBST(0,0)' →LST      @Turn the dummy subst to a list
1 4 ROLL PUT           @Put expression in position 1
2 ROT PUT              @and substitution equation in 2
→ALG                   @Turn it to algebraic object.
```

I like →LST, and →ALG very much. Nonetheless the usage of such tricks just to enter something that is syntactically completely correct, shouldn't be necessary. Another trick would be to store the substitution equation in some variable. For example, enter X = π and store it in variable Y. Then enter SUBST(DERIV(SIN(X),X),T). This expression in possible also from the command line since the second argument of SUBST doesn't contain a "=". If you expand, you get the correct result −1. Not so much of a trick here, but still an unnecessary complication. Purge now Y.

And what happens with |? We start again with RPL syntax. Go to the EQW and enter DERIV(SIN(X),X). Enter the list {X 'π}. Be careful to enter π in single quotes in the list. This is the other method to ensure that π is an algebraic object in the list and not just the command π. (We already used another method, namely entering X, π, and then constructing the list by entering 2 and pressing →LIST.) Now, press ⬚ to get COS(π), which isn't completely expanded but correct. Let's try that in algebraic syntax. Go to the EQW and enter:

$\text{DERIV}(\text{SIN}(X),X)|_{X=\pi}$

Expand that to get the correct result, -1.

And what about the way that was inherited from the HP48? Can we store a value in some variable and find the derivative for this variable? For example, store π in variable T. Enter $\text{SIN}(T)$ and then 'T' in quotes. Press DERIV. You get $\text{COS}(T)$ which you can expand to get -1. Go to the EQW and type $\text{DERIV}(\text{SIN}(T),T)$. Put that on the stack and expand. Now you get an error Bad Argument Value. EVAL doesn't work either and you lose the last argument since you get 0, π, and the command DERIV on the stack. If you want to get the correct result you have to use QUOTE. Enter $\text{DERIV}(\text{SIN}(\text{QUOTE}(T)),\text{QUOTE}(T))$ and expand to get -1. But note again that if you have for example X as the current variable, store π in X, then enter $\text{DERIV}(\text{SIN}(\text{QUOTE}(X)),\text{QUOTE}(X))$ and expand, then you get the question to purge the current variable. If you choose "No", then the operation errors out with Mode Switch Cancelled. If you choose "Yes", then X is purged and the calculation returns $\text{COS}(X)$, because the variable X doesn't exist any more. Instead of expanding in this case you can press EVAL twice, to avoid the question about purging the current variable.

We continue examining the behaviour of DERIV when we take derivatives of expressions with variables that themselves contain other expressions (just as we did with ∂). If you still have the following variables, then you don't need to re-create them. We need:

X^2 in R

$\text{SIN}(Y)$ in S

$S^2 - R$ in U.

```
<< → X 'X^3/(X-1)' >>
```
in V

```
<< → R 'R*(R+1)' >>
```
in W

```
<< → R X '(R-1)/X' >>
```
in Z

```
<< → S 'S/R' >>
```
in Q

We also need:

```
<< → X
  <<  0 1 3 FOR I
        X I R→I ^ +
      NEXT
  >>
>>
```

in Z1.

And last thing:

```
<< → R X
  <<  R EXP S EXP + X / >>
>>
```

in Z2.

Enter 'R' (with quotes) and press ENTER to make copy it on stack level 2. Press DERIV to take the derivative of R for R. The result is 1, which shows that DERIV (like ∂) didn't care about the fact that X^2 is stored in R. If you enter 'R' and then X, and take the derivative, then you get a 0, which again shows: When some expression is on stack level 2 and we take its derivative for some of its variables using the command DERIV, then derivation is carried out without first evaluating that variable. This is good for finding slopes of functions by storing

something in the variable for which we take the derivative and then using DERIV, as we already saw on the previous page. Another example of this case: Enter S^2, then 'S' (with quotes) and press DERIV to get $2\ S$. If you enter $\text{DERIV}(S^2,S)$ from the EQW and press EVAL, you will get $2\ \text{SIN}(Y)\ \text{COS}(Y)$. If you expand instead of evaluating, then you get $2\ \text{COS}(Y)\ \text{SIN}(Y)$, which is the same but with the factors reordered. Obviously, expanding or evaluating DERIV(expression,variable), when expression contains variables where other expressions are stored, will first differentiate and *then* replace the variables with their contents.

If you want to first evaluate some variable contained in an expression, and then take the derivative with DERIV, you have to evaluate first. For example, enter U^2. We stored S^2-R in U, SIN(Y) in S, and X^2 in R. If you want to take the derivative of the evaluated expression U^2 for X, then you have to press EVAL first. This completely evaluates U^2 to $\text{SIN}(Y)^4-2\ X\ \text{SIN}(Y)^2+X^4$. Now you can enter X, press DERIV and then EXPAND, to get the fully expanded expression $-\left(4\ X\ \text{SIN}(Y)^2-4\ X^3\right)$.

And what about taking the derivative of the *partially* evaluated U^2 for S? We try with SHOW again. Enter U^2, then 'S' (in quotes) and press SHOW, to stop evaluation when the variable S is shown. This will result in $\left(S^2-R\right)^2$, allowing us to enter 'S' and take the derivative pressing DERIV.

Now, what if the expression, which we take the derivative of, is a user defined function? Let's take a look at that. Enter $V(X)^2$ and then X. If you now press DERIV then the derivation takes a bit longer, and returns:

$$2\ \frac{X^3}{X-1}\ \frac{(X-1)\ 3\ X^2-X^3}{\text{SQ}(X-1)}$$

This shows: Since V is a user defined function, $V(X)^2$ is *first* evaluated to

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

and *then* the differentiation for X is carried out! Exactly the same result will be returned if you enter $\text{DERIV}\left(V(X)^2,X\right)$ and then evaluate. If you enter $\text{DERIV}\left(V(X)^2,X\right)$ and expand, then you get the result:

$$\frac{4\ X^6-6\ X^5}{X^3-3\ X^2+3\ X-1}$$

which is the same as before, but completely expanded. If you enter $\text{DERIV}\left(V(X)^2,X\right)$ and then press FACTOR or COLLECT, nothing happens and the expression remains unchanged on stack level 1. Like in case of ∂ this allows to collect unevaluated differential forms, which for example can be used to bring differential equations to a much more readable form. For example, enter the expression $\text{DERIV}\left(V(X)^2,X\right)\ X+\text{DERIV}\left(V(X)^2,X\right)\ A$ and press COLLECT, to get $(X+A)\ \text{DERIV}\left(V(X)^2,X\right)$. If you want the opposite to happen, then don't press EVAL or EXPAND, but DISTRIB. If you press DISTRIB with the last result still on stack level 1, then you will get $X\ \text{DERIV}\left(V(X)^2,X\right)+A\ \text{DERIV}\left(V(X)^2,X\right)$. And what about transforming $\text{DERIV}\left(V(X)^2,X\right)$ to $2\ V(X)\ \text{DERIV}(V(X),X)$, or some equivalent form? Can we use step by step mode? Let's see. Switch to

step by step mode and enter $\mathrm{DERIV}(V(X)^2,X)$. If you evaluate or expand, then you see that… it doesn't work! You just get the same results like in non step by step mode. Step by step seems to work better with ∂ (as we already have seen) in this case. Deactivate the step by step feature now.

Let's see now what happens when we take the derivative of some user defined function, to which we have given a variable as argument, that itself contains some expression. Enter the expression $V(R)$. Now, enter 'R' and press DERIV. The result is 0 because the user defined function *is evaluated before derivation*. This means that $V(R)$ was first evaluated to:

$$\frac{R^3}{R-1}$$

Then the variable R in this expression was evaluated, which returned:

$$\frac{X^6}{X^2-1}$$

since R contains X^2. It was this expression of which the derivative for R was taken. And since the expression

$$\frac{X^6}{X^2-1}$$

doesn't depend on R, the derivative was 0. If you enter $V(R)$ again, then enter X and press DERIV, you get the result:

$$\frac{\left(X^2-1\right)\ 6\ X^5-X^6\ 2\ X}{\mathrm{SQ}\left(X^2-1\right)}$$

the derivative of the completely evaluated user defined function $V(R)$ for variable X. Notice here that the variable of derivation, R is not getting evaluated to X^2.

Now the same in algebraic syntax. Enter $\mathrm{DERIV}(V(R),R)$. If you evaluate this, then the result is again:

$$\frac{\left(X^2-1\right)\ 6\ X^5-X^6\ 2\ X}{\mathrm{SQ}\left(X^2-1\right)}$$

which is the derivative of the user defined function $V(R)$, where the argument R has been evaluated to X^2, but also the variable of derivation R has been evaluated to X^2. But let's see what happened stepwise. In algebraic expressions, in general the innermost nested things get evaluated first. In $\mathrm{DERIV}(V(R),R)$ the argument of the function V was first evaluated to X^2. Then the function V took X^2 and returned:

$$\frac{X^6}{X^2-1}$$

The variable of derivation, R, was also evaluated to X^2. So the expression was converted to:

$$\mathrm{DERIV}\left(\frac{X^6}{X^2-1},X^2\right)$$

Here we must watch out! When the command DERIV takes the derivative not for a single variable, but for an expression, then… it takes the derivative for the first variable that it finds in the expression, for which it takes the derivative. That means, that the returned result is the result of the differentiation:

$$\frac{\partial}{\partial X}\left(\frac{X^6}{X^2-1}\right)$$

and not the result of the differentiation:

$$\frac{\partial}{\partial\left(X^2\right)}\left(\frac{X^6}{X^2-1}\right)$$

If you enter:

$$\frac{X^6}{X^2-1}$$

then X, and then press DERIV, you get the result:

$$\frac{\left(X^2-1\right)\ 6\ X^5-X^6\ 2\ X}{\mathrm{SQ}\left(X^2-1\right)}$$

If you enter again:

$$\frac{X^6}{X^2-1}$$

then X^2 and press DERIV again, you get again:

$$\frac{\left(X^2-1\right)\ 6\ X^5-X^6\ 2\ X}{\mathrm{SQ}\left(X^2-1\right)}$$

The same result as for taking the derivative for X was returned. Expand it to get:

$$\frac{4\ X^7-6\ X^5}{X^4-2\ X^2+1}$$

Our program dF1F2 fed with $\frac{X^6}{X^2-1}$ and X^2, returns the result:

$$\left\{\frac{2\ X^6-3\ X^4}{X^4-2\ X^2+1}\quad\frac{2\ X^6-3\ X^4}{X^4-2\ X^2+1}\right\}$$

which shows that:

$$\frac{\partial}{\partial\left(X^2\right)}\left(\frac{X^6}{X^2-1}\right)=\frac{2\ X^6-3\ X^4}{X^4-2\ X^2+1}.$$

If you have problems to understand what DERIV does in such cases, do the following: Enter X^2, then X, and press DERIV to get $2\ X$, which is OK. But now, enter X^2, then X^2, and press DERIV to get $2\ X$ again, which in this case is wrong, since

$$\frac{\partial X^2}{\partial\left(X^2\right)}=1$$

Another example: Enter $X\ Y$, then X and press DERIV to get Y, which is OK. But if you enter $X\ Y$, then $X+Y$, and press DERIV, you get Y again, which is wrong. The command just took the derivative for the first variable in $X+Y$, which is X. And if you enter $X\ Y$, then $Y+X$, and then press DERIV, you get X, which is also wrong. In this case the derivative was taken for Y, because it was the first variable in $Y+X$. The problem here is that DERIV allows expressions to be written, where the variable of derivation is itself some expression, but it differentiates for the first variable in this "expression of derivation", and returns wrong results. So, when it comes to such derivatives we have to use a program. For the time being use dF1F2 as it is now, though it will

also have its problems when taking derivatives for expression that contain more than one variable, like for example

$$\frac{\partial X\ Y}{\partial(X+Y)}$$

We will make it better later on.

We try now to let $V(R)$ be evaluated up to

$$\frac{R^3}{R-1}$$

and then take the derivative, without replacing R by its contents. We have to use the command QUOTE again. Go to the EQW and enter $V(\text{QUOTE}(R))$. Enter 'R' (in quotes) and press DERIV to get:

$$\frac{(R-1)\ 3\ R^2-R^3}{\text{SQ}(R-1)}$$

This is the derivative of the user defined function $V(R)$ evaluated up to

$$\frac{R^3}{R-1}$$

To get the fully expanded form without evaluating all occurrences of R to X^2, you can enter $V(\text{QUOTE}(\text{QUOTE}(R)))$, then 'R', then press DERIV to get:

$$\frac{(\text{QUOTE}(R)-1)\ 3\ \text{QUOTE}(R)^2\ \text{d1QUOTE}(R)-\text{QUOTE}(R)^3\ \text{d1QUOTE}(R)}{\text{SQ}(\text{QUOTE}(R)-1)}$$

If you evaluate now, you will get:

$$\frac{2\ R^3-3\ R^2}{R^2-2\ R+1}$$

which is the fully expanded result that still contains Rs and not Xs.

If you want to enter the derivative for R in algebraic syntax, then you have to QUOTE not only the variable R that appears as argument of the user defined function V, but also the variable of derivation. You enter $\text{DERIV}(V(\text{QUOTE}(R)),\text{QUOTE}(R))$, to control evaluation and let the expression be converted to:

$$\text{DERIV}\left(\frac{R^3}{R-1},R\right)$$

instead of:

$$\text{DERIV}\left(\frac{R^3}{R-1},X^2\right)$$

If you evaluate the expression $\text{DERIV}(V(\text{QUOTE}(R)),\text{QUOTE}(R))$, then the result is:

$$\frac{(R-1)\ 3\ R^2-R^3}{\text{SQ}(R-1)}$$

But if you would evaluate the expression $\text{DERIV}(V(\text{QUOTE}(R)),R)$, you would get 0 because it would be evaluated to:

$$\text{DERIV}\left(\frac{R^3}{R-1},X^2\right)$$

Enter $W(X)$ and then X. Press DERIV to get the result $X+1+X$. The user defined function $W(X)$ was first evaluated, and that returned $X\ (X+1)$. Then the derivative for X was found, $X+1+X$. (If you enter X and press the menu key [W], the the result is $X\ (X+1)$. Entering $\mathrm{DERIV}(W(X),X)$ and pressing EVAL returns the same result. If you would have pressed EXPAND instead of EVAL, then the result would be $2\ X+1$, the completely expanded form.

If you enter $W(R)$, then 'R' and then press DERIV, the result is 0, because: The user function $W(R)$ was first evaluated and returned $X^2\ (X^2+1)$. Then the derivative of this expression for R was taken, which was found to be 0, because the expression didn't contain R. But if you evaluate $\mathrm{DERIV}(W(R),R)$, then the result is $2\ X\ (X^2+1)+X^2\ 2\ X$. Remember, evaluation of algebraic objects, includes evaluation of user defined functions and variables. So the evaluation went the way:

1) $\mathrm{DERIV}(W(R),R)$
2) $\mathrm{DERIV}(W(X^2),X^2)$
3) $\mathrm{DERIV}(X^2\ (X^2+1),X^2)$

Then, because DERIV had an expression (X^2) and not a single variable to differentiate for, it differentiated for the first variable of this expression (X), and found $2\ X\ (X^2+1)+X^2\ 2\ X$.

If you enter 'R', press W, then enter 'R' again and then press DERIV, the result will be $R+1+R$. In RPL syntax no evaluation of the global variable R took place. So, we get the result $R\ (R+1)$ when we apply the user defined function W on the argument 'R' which is on the stack. After this there is no user defined function that has to be evaluated before taking the derivative. The expression $R+1+R$ doesn't contain any user defined function, and so DERIV simply takes the derivative of this expression.

If you enter $W(\mathrm{QUOTE}(R))$, then 'R' and then press DERIV, the result will be $R+1+R$. In algebraic syntax, if you enter $\mathrm{DERIV}(W(\mathrm{QUOTE}(R)),\mathrm{QUOTE}(R))$ and press EVAL, the HP49G returns the result $2\ R+1$. Both results are what we expect to get according to what we know until now.

If you enter $W(R)$, then X, and then press DERIV, then the result is $2\ X\ (X^2+1)+X^2\ 2\ X$. This shows again, that $W(R)$ was evaluated the way we already know, which returned $X^2\ (X^2+1)$, and then the derivative of this result for X was found. The same result is found if you evaluate $\mathrm{DERIV}(W(R),X)$.

Now we nest some user defined functions. For example, enter $W(V(X))$, enter X and then press DERIV to get the expression:

$$\frac{(X-1)\ 3\ X^2-X^3}{\mathrm{SQ}(X-1)}\left(\frac{X^3}{X-1}+1\right)+\frac{X^3}{X-1}\ \frac{(X-1)\ 3\ X^2-X^3}{\mathrm{SQ}(X-1)}$$

Exactly the same would happen, if we entered $\mathrm{DERIV}(W(V(X)),X)$ and evaluated.

Now we do the same using variable R as argument. If you enter 'R', press V, press W, enter 'R' again, and press DERIV, you find:

$$\frac{(R-1)\ 3\ R^2-R^3}{\mathrm{SQ}(R-1)}\left(\frac{R^3}{R-1}+1\right)+\frac{R^3}{R-1}\ \frac{(R-1)\ 3\ R^2-R^3}{\mathrm{SQ}(R-1)}$$

If you enter W(V(R)), then 'R' and then press DERIV, then you find 0. In the first case, the HP49G used the argument which we gave it, namely 'R' and just applied the user functions V and W on this argument. Since the result didn't contain any user defined functions any more, the function DERIV just differentiated what it found on the stack, without any evaluation. In the second case however, the command DERIV found an algebraic expression that contained user defined functions, so it evaluated them first, found:

$$\frac{X^6}{X^2-1}\left(\frac{X^6}{X^2-1}+1\right)$$

and so the differentiation for R returned 0.

We continue on user defined functions with RPL definitions that return an algebraic object. Enter DERIV(Z1(X),X) and expand to get 3 X^2+2 X+1. Alternatively you can also enter Z1(X), then X, and then press DERIV to find the derivative.

If we evaluate DERIV(Z1(R),R) we get 2 X+4 X^3+6 X^5. Why the difference? It seems that evaluating the above, triggers first evaluation of the user defined function Z1 with argument R, which results in $R+R^2+R^3$. But then R is also evaluated and this results in $X^2+\left(X^2\right)^2+\left(X^2\right)^3$. The variable of derivation is also evaluated, and so the derivative is transformed to $\text{DERIV}\left(X^2+\left(X^2\right)^2+\left(X^2\right)^3,X^2\right)$. As we know, DERIV finds then the derivative for X and not for X^2. This way we come to the result 2 X+4 X^3+6 X^5. Notice how this differs from evaluation of the analogous expression $\frac{\partial}{\partial R}(Z1(R))$.

In RPL syntax, if we enter Z1(R), then 'R' and press DERIV, the result is 0. EVALuate Z1(R) and take the derivative for 'R' to understand why. What will be returned if you take the derivative of Z1(QUOTE(R)) for 'R'?

The command DERIV can also be used for carrying out formal derivations. Enter F(X) and then X, and press DERIV to get the formal derivative d1F(X), which is returned because the function F depends on X but the HP49G doesn't know *how* it depends on X. Enter F(X(T)) then T and then press [DERIV], to get d1X(T) d1F(X(T)), exactly just like if you had used ∂ instead.

But there are also differences to ∂. Since DERIV accepts also expressions as variables of differentiation, we can enter F(X(T)), then X(T), and then press DERIV. We can do this and we get the result d1X(T) d1F(X(T)), where d1F(X(T)) means the derivative of F for X(T).

We also see that the command DERIV can't be used for finding such things like for example

$$\frac{\partial \text{SIN}(X)}{\partial \text{SIN}(X)}$$

right out of the box (simply because it returns the wrong result). If we enter DERIV(T,T), then T = SIN(X), and then press SUBST, the HP49G returns DERIV(SIN(X),SIN(X)), which isn't much of a help, since the evaluation of this expression will still return COS(X) and not 1. But entering DERIV(T,T), then entering the list {T SIN(X)}, and then pressing ▫, does return the correct result 1. This works also in algebraic syntax. Enter $\text{DERIV}(T,T)\big|_{T=\text{SIN}(X)}$ and evaluate to get again 1.

Is that a hope for easy formulation and correct evaluation of arbitrary

$$\frac{\partial \text{expression1}}{\partial \text{expression2}} \ ?$$

Might be. Let's try some more complicated examples, and see if we get the correct results.

Suppose we want to find:

$$\frac{\partial X^2-3\ X+3}{\partial\left(2\ X^2\right)}$$

We can't enter directly $\text{DERIV}\left(X^2-3\ X+3,2\ X^2\right)$, because this would actually return the result of $\text{DERIV}\left(X^2-3\ X+3,X\right)$. But if we think the expression of derivation $2\ X^2$ as a single variable, say T, then we have:

$$2\ X^2=T \Leftrightarrow X^2=\frac{T}{2}\ ,\quad 2\ X^2=T \Leftrightarrow X^2=\frac{T}{2} \Leftrightarrow X=\pm\sqrt{\frac{T}{2}}$$

So we can write the derivative as:

$$\text{DERIV}\left(\frac{T}{2}-3\ \sqrt{\frac{T}{2}}+3,T\right)$$

or:

$$\text{DERIV}\left(\frac{T}{2}+3\ \sqrt{\frac{T}{2}}+3,T\right)$$

We try with the first expression. Enter:

$$\text{DERIV}\left(\frac{T}{2}-3\ \sqrt{\frac{T}{2}}+3,T\right)$$

then enter the list $\{T\quad 2\ X^2\}$ and press ▶ to get:

$$\frac{\left(2\ |X|-3\right)\ |X|}{\text{SQ}(X)\ \ 4}$$

which is OK. From the second derivative we get

$$\frac{\left(2\ |X|+3\right)\ |X|}{\text{SQ}(X)\ \ 4}$$

The program dF1F2 finds exactly the same results, but it also does the above parametrisation automatically. The advantage of using DERIV with | is that the derivation and the back substitution are carried out in one step. So we keep this in mind as it might prove useful for our improvements of dF1F2. The real problem comes in cases like for example:

$$\frac{\partial(X+Y)^2+X}{\partial X+Y}$$

that is when we derivative for an expression that contains more than one variables. In such cases, if we do the parametrisation $X+Y=T$, then writing $\text{DERIV}\left(T^2+T-Y,T\right)$ would be wrong (for the CAS logic of the HP49G). We should write $\text{DERIV}\left(T^2+T-Y(T),T\right)$, to denote that the remaining Y, still is a function of the derivation variable T. But then, if we enter $\{T\quad X+Y\}$ and press ▶, we get $2\ (X+Y)+1-\text{d1Y}(X+Y)$. The last term in this expression, $\text{d1Y}(X+Y)$, denotes the derivative of Y for $X+Y$, i.e. a function Y

that depends on… the sum $X+Y$, i.e. on itself!!! This is apparently somehow pathological. But if we remember that $X+Y=T$, we see that $d1Y(X+Y)=d1Y(T)$. How to avoid such apparently pathological results on the HP49G? Remember, $d1Y(X+Y)$ is the result of using the list $\{T\ \ X+Y\}$ as argument for $|$. In the expression $T^2+T-Y(T)$ which must be differentiated, Y is a function of T. But in the list $\{T\ \ X+Y\}$ the quantity Y doesn't depend on anything, it is a free variable. We should rather have used $\{T\ \ X+Y(T)\}$. Enter $DERIV(T^2+T-Y(T),T)$, then $\{T\ \ X+Y(T)\}$ and then press $|$. The result is $2\ (X+Y(T))+1-d1Y(X+Y(T))$. Now the formal derivative $d1Y(X+Y(T))$ is no more so pathological, because since $T=X+Y$, we have $Y=T-X$, and so for the derivative $d1Y(X+Y)$ we have $d1Y(X+Y(T))=d1Y(X+T-X)=d1Y(T)$. We only had to make the dependence of Y on T explicit by writing $Y(T)$.

Anyway, we see that taking derivatives for expressions and not for single variables is not always so straight forward. We will return to them, as already said, when we see how to take derivatives of functions in parametric representation.

We go on examining formal derivatives and DERIV. Enter $F(S)$, then Y, and press DERIV to find the derivative. The result is $COS(Y)\ d1F(SIN(Y))$. The HP49G used the chain rule to return the product of the derivative for Y of the inner function $SIN(Y)$, which is stored in S, with the derivative of the outer function. As the latter is totally undefined, the HP49G returns $d1F(SIN(Y))$, to denote that derivative in a general abstract way. If we evaluate or expand $DERIV(F(S),Y)$, we get $COS(Y)\ d1F(SIN(Y))$ again. Notice here the difference to ∂, which returned 0. In both cases the command DERIV triggered evaluation of the variable S in $F(S)$, and the result was $F(COS(Y))$. Then, the derivation for Y was carried out, and $COS(Y)\ d1F(SIN(Y))$ was found. At this point, we collect all the cases in a single table again, for a better understanding of the behaviour of the command DERIV. There are four tables for this. The first table is on the next page and contains cases of differentiation of an expression or a user defined function for a variable. The second table is on the page after the next and contains cases of differentiation of an expression or a user defined function for a quoted variable a' la $QUOTE(variable)$. The third table is on the page 1-45 and contains the cases of differentiation of a function that isn't defined, that is cases of formal differentiation for a variable. The fourth table is on the page 1-46 and contains the cases of differentiation of a function that isn't defined, that is cases of formal differentiation for a quoted variable a' la $QUOTE(variable)$. In all tables, regions with the same colour are those which return their results using the same mechanism. Like in the tables for ∂, it is not the same result that makes up a region, but rather the same *way* that is followed by the HP49G to return these results.

Let's do some examples that are more complex. We use what we know until now, to explain how the results are derived. Keep all variables and user defined functions because the following examples use them.

Enter $F(V(X))$, then enter 'R' (in quotes) and press DERIV. The result is 0, as we expect, because as we have seen already, the user defined function $V(X)$ (argument of F) is evaluated before differentiation. This gives

$$F\left(\frac{X^3}{X-1}\right)$$

Now, the HP49G sees that this function, though undefined, doesn't depend on R, because R doesn't appear as an argument of F. As the derivation variable R isn't evaluated the derivative is found to be 0.

	RPL Syntax: Enter expression then variable, then use DERIV		Algebraic Syntax: Enter DERIV(Expression,Var), then evaluate or expand	
Expression \ Variable of derivation	'R'	'X'	'R'	'X'
R^2+R (X^2 stored in R)	$2\ R+1$	0	$4\ X^3+2\ X$	$4\ X^3+2\ X$
V(X) (User function V defined as: << -> X 'X^3/(X-1)' >>)	0	$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}$	$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}$	$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}$
V(QUOTE(X)) (User function V defined as: << -> X 'X^3/(X-1)' >>)	0	$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}$	$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}$	$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}$
V(R) (User function V defined as: << -> X 'X^3/(X-1)' >>, X^2 stored in R)	0	$\frac{(X^2-1)\ 6\ X^5-X^6\ 2\ X}{SQ(X^2-1)}$	$\frac{(X^2-1)\ 6\ X^5-X^6\ 2\ X}{SQ(X^2-1)}$	$\frac{(X^2-1)\ 6\ X^5-X^6\ 2\ X}{SQ(X^2-1)}$
V(QUOTE(R)) (User function V defined as: << -> X 'X^3/(X-1)' >>, X^2 stored in R)	$\frac{(R-1)\ 3\ R^2-R^3}{SQ(R-1)}$	0	0	0

No evaluation of expression and derivation variables before derivation.	Evaluation of expression and derivation variables before derivation.	Evaluation of function and function arguments but not of derivation variable before derivation.	Evaluation of function, function arguments and derivation variable before derivation.	Evaluation of function but not of function arguments and derivation variable before derivation.	Evaluation of function and derivation variable but not of function argument before derivation.

	RPL Syntax: Enter expression then variable, then use DERIV		Algebraic Syntax: Enter DERIV(Expression,Var), then evaluate or expand	
Expression \ Variable of derivation	'QUOTE(R)'	'QUOTE(X)'	'QUOTE(R)'	'QUOTE(X)'
R^2+R (X^2 stored in R)	----------	----------	0	$4\ X^3+2\ X$
V(X) (User function V defined as: << -> X 'X^3/(X-1)' >>)	----------	----------	0	$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}$
V(QUOTE(X)) (User function V defined as: << -> X 'X^3/(X-1)' >>)	----------	----------	0	$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}$
V(R) (User function V defined as: << -> X 'X^3/(X-1)' >>, X^2 stored in R)	----------	----------	0	$\frac{(X^2-1)\ 6\ X^5-X^6\ 2\ X}{SQ(X^2-1)}$
V(QUOTE(R)) (User function V defined as: << -> X 'X^3/(X-1)' >>, X^2 stored in R)	----------	----------	$\frac{(R-1)\ 3\ R^2-R^3}{SQ(R-1)}$	0

DERIV Error: Bad argument value

Evaluation of function and function arguments but not of derivation variable before derivation.

Evaluation of function but not of function arguments and derivation variable before derivation.

If you enter $F(V(X))$ again, but then take the derivative with respect to X using the command DERIV, then the result is:

$$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}\quad d1F\left(\frac{X^3}{X-1}\right)$$

Following again our known rules we can understand that. First of all the function $V(X)$ is evaluated. The result of this action is:

$$\frac{X^3}{X-1}$$

This is used as argument for the undefined function F, that is we get:

$$F\left(\frac{X^3}{X-1}\right)$$

This result is then differentiated for X. According to the rules of derivation we get:

$$\frac{(X-1)\ 3\ X^2-X^3}{SQ(X-1)}\quad d1F\left(\frac{X^3}{X-1}\right)$$

where the expression

	RPL Syntax: Enter expression then variable, then use DERIV		Algebraic Syntax: Enter DERIV(Expression,Var), then evaluate		
Expression \ Variable of derivation	'R'	'X'	'R'	'X'	
$F(X)$	0	$d1F(X)$	$d1F(X)$	$d1F(X)$	Formal differentiation with expansion of intermediate differential forms.
$F(X^2)$	0	$2\ X\ d1F(X^2)$	$2\ X\ d1F(X^2)$	$2\ X\ d1F(X^2)$	Formal differentiation with expansion of intermediate differential forms but no evaluation of the derivation variable.
$F(QUOTE(X))$	0	$d1F(X)$	$d1F(X)$	$d1F(X)$	
$F(R)$ X^2 stored in R	0	$2\ X\ d1F(X^2)$	$2\ X\ d1F(X^2)$	$2\ X\ d1F(X^2)$	Formal differentiation with expansion of intermediate differential forms but no evaluation of function arguments and derivation variable.
$F(R^2)$ X^2 stored in R	0	$4\ X^3\ d1F(X^4)$	$4\ X^3\ d1F(X^4)$	$4\ X^3\ d1F(X^4)$	
$F(QUOTE(R))$ X^2 stored in R	$d1F(R)$	0	$2\ X\ d1F(X^2)$	$2\ X\ d1F(X^2)$	In algebraic syntax quoting arguments of functions of which we take derivatives, seems not to have any effect at all

$d1F\left(\frac{X^3}{X-1}\right)$ stands for the formal derivative of

$$F\left(\frac{X^3}{X-1}\right)$$

that means a derivative which can't be explicitly given, since F is undefined.

We try to find the derivative of $V(F(R))$ for R and for X. If you enter $V(F(R))$, then 'R', and then press DERIV, get 0. This shows again, that the argument of F, which is R, gets evaluated to its contents, which are X^2. After this we have $V(F(X^2))$, which differentiated for R must return 0, since it doesn't depend on R. But if you enter $V(F(R))$, then X, and then press DERIV, you get:

$$\frac{(F(X^2)-1)\ 3\ F(X^2)^2\ 2\ X\ d1F(X^2) - F(X^2)^3\ 2\ X\ d1F(X^2)}{SQ(F(X^2)-1)}.$$

Let's see how this was produced. First, the argument of F was evaluated. This argument was R, and since X^2 is stored in R, the result of the evaluation was X^2. This was used as argument for F, and because F is undefined, we simply get $F(X^2)$. This expression was

	RPL Syntax: Enter expression then variable, then use DERIV		Algebraic Syntax: Enter DERIV(Expression,Var), then evaluate	
Expression \ Variable of derivation	'QUOTE(R)'	'QUOTE(X)'	'QUOTE(R)'	'QUOTE(X)'
$F(X)$	----------	----------	0	$d1F(X)$
$F(X^2)$	----------	----------	0	$2\ X\ d1F(X^2)$
$F(QUOTE(X))$	----------	----------	0	$d1F(X)$
$F(R)$ X^2 stored in R	----------	----------	0	$2\ X\ d1F(X^2)$
$F(R^2)$ X^2 stored in R	----------	----------	0	$4\ X^3\ d1F(X^4)$
$F(QUOTE(R))$ X^2 stored in R	----------	----------	0	$2\ X\ d1F(X^2)$

DERIV Error: Bad Argument Value

Formal differentiation with expansion of intermediate differential forms but no evaluation of the derivation variable.

In algebraic syntax quoting arguments of functions of which we take derivatives, seems not to have any effect at all

used as argument for V. So instead of getting:

$$\frac{X^3}{X-1}$$

we got:

$$\frac{F(X^2)^3}{F(X^2)-1}$$

Next, the differentiation for X was carried out. According to the rules of differentiation for a ratio, we have:

$$\frac{\partial}{\partial X}\left(\frac{F(X^2)^3}{F(X^2)-1}\right)=\frac{\left(F(X^2)-1\right)\ \frac{\partial}{\partial X}\left(F(X^2)^3\right)-F(X^2)^3\ \frac{\partial}{\partial X}\left(F(X^2)-1\right)}{SQ\left(F(X^2)-1\right)}$$

Carrying out the derivations of the right hand side of the last equation, and keeping in mind that $d1F(X^2)$ is the formal derivative of $F(X^2)$, we see that the result returned by the HP49G was correct (inside the frame of its evaluation rules).

Do some examples for yourself and try to predict the behaviour of the command DERIV, using the knowledge that we have so far. Remember that QUOTE can also be used with formal functions, which means that F(QUOTE(R)) is perfectly OK. Note that the command DERIV (and also the function ∂) doesn't like QUOTE(variable) as the differentiation variable in RPL syntax. But DERIV accepts QUOTE(variable) as the differentiation variable in algebraic syntax. Note also, that in algebraic syntax, quoting arguments of functions doesn't seem to work at all. The arguments seem to be always completely evaluated before differentiation.

Take heart, we almost finished this exhaustive study of the derivation commands. We examine some more complex examples of derivation of undefined functions with DERIV. Enter DERIV(F(Y,X),X) and expand. This will return d1F(Y,X), just as expected. Enter DERIV(F(X) G(X),X) and expand this to get:

$$G(X)\ d1F(X)+F(X)\ d1G(X)$$

Enter:

$$\text{DERIV}\left(\frac{F(X)\ G(X)^2}{G(X)+X},X\right)$$

If you now expand you get:

$$\frac{\left(G(X)^3+X\ G(X)^2\right)\ d1F(X)+\left(G(X)^2+2\ X\ G(X)\right)\ F(X)\ d1G(X)-G(X)^2\ F(X)}{G(X)^2+2\ X\ G(X)+X^2}$$

This contains only derivatives of a single function for a single variable, which makes easier to see how the differential forms participate to built-up the derivative

$$\frac{\partial}{\partial X}\left(\frac{F(X)\ G(X)^2}{G(X)+X}\right).$$

We see that DERIV behaves like ∂ when it finds some function of which it doesn't know how to built the derivative. Also, exactly like working with ∂, the expressions DERIV(G(X²),X) and d1G(X²) are not equal. In this particular case for the HP49G the relation holds:

$$\text{DERIV}\left(G(X^2),X\right)=2\ X\ d1G(X^2)$$

Only in cases where the function has simple names as arguments, like for example $G(X,Y)$, the two notations $\text{DERIV}(G(X,Y),X)$ and $d1G(X,Y)$ are equivalent.

The command DERIV can also be used for implicit derivations in the same syntax like ∂. Enter $\text{SIN}(Y(X)) = Y(X) - X$, then enter X, and then press DERIV, to get $\text{COS}(Y(X))\ d1Y(X) = d1Y(X) - 1$.

Last thing we are going to examine is how substitutions behave, when used with expressions that contain DERIV. Enter $\text{DERIV}(X^2 - 1, X)$, then $X = Y$ and then press SUBST. The result is $\text{DERIV}(Y^2 - 1, Y)$, which shows that SUBST only substituted all occurrences of X with Y but didn't do anything else. Undo the last operation, and edit the equation in stack level 1 to $X = Y^2$. Press again SUBST. This time the HP49G returns $\text{DERIV}\left(\left(Y^2\right)^2 - 1, Y^2\right)$, which as we know is exactly the same as if we had entered $\text{DERIV}\left(\left(Y^2\right)^2 - 1, Y\right)$.

On the other hand, we have also the command | for substitutions. Undo the last operation, drop the equation $X = Y^2$, and enter the list $\{X\ \ Y\}$. Press |. The result is now the expression $2\ \ Y$. The function | not only did the substitution but triggered also the CAS to carry out the derivation, after the substitution. Undo the operation, and edit the list to $\{X\ \ Y^2\}$. Press | again to get $2\ \ Y^2$, which as we have seen is the correct result of

$$\frac{\partial}{\partial Y^2}\left(\left(Y^2\right)^2 - 1\right)$$

Strange is only that if you enter directly $\text{DERIV}\left(\left(Y^2\right)^2 - 1, Y^2\right)$ and expand, you are going to get $4\ \ Y^3$, which is wrong because it is the result of derivation for Y and not for Y^2. You get the right result too, if you enter $\text{DERIV}\left(X^2 - 1, X\right)\Big|_{X=Y^2}$ and expand or evaluate.

Another available command for derivation is the command DERVX. It works like DERIV, but it always takes the derivative for the current VX. It needs only one argument, namely the expression of which we want to find the derivative.

From what we have seen until now, it seems that EVAL gives us more detailed control of what and how we want to work with derivatives. Especially in combination with QUOTE and SHOW, this command is very flexible. And this makes it a very powerful tool for programming, when we want to do something special exactly the way we want to. On the other hand, the real power of EXPAND is that it is more "fire and forget", as it (almost) always does its job from the beginning to the end without any user intervention. (Goodness! I talk like a military man - too much TV-war in the last days.)

Now that we have finished the long path of the workings of derivation commands, we return to our main path. With the HP49G it is easy not only to find some derivative or slope of a given expression, but also to find if some expression has a derivative at some given point. Consider for example X^2. How can we find out if this expression has a derivative at $X = X_0$? In this case it is a piece of cake. One of the many many ways to do that is to enter

$$\frac{\partial}{\partial X}\left(X^2\right)\Big|_{X=X_0}$$

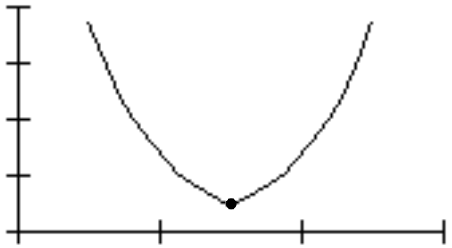
and expand. We get $2\ \ X_0$ which is defined everywhere. The derivative exists at any point X_0. Another example: Does the expression $\sqrt[3]{X}$ have a derivative at $X = 0$? If we enter:

$$\frac{\partial}{\partial X}\left(\sqrt[3]{X}\right)\Big|_{X=0}$$

and expand, then we get ?. The derivative isn't defined at $X = 0$. Same with $e^{|X-2|}$ at $X = 2$. Enter:

$$\frac{\partial}{\partial X}\left(e^{|X-2|}\right)\Big|_{X=2}$$

and expand to get ?, which shows that the derivative is not defined at that point. Especially for the last example, $e^{|X-2|}$, we can use the program ISCONT? of SESELIMA to check if it is continuous at $X = 2$. If you want to check that, enter $e^{|X-2|}$ and $X = 2$ again and let ISCONT? run. The result will be a 1, which shows that the expression is continuous at that point. If you plot the function, then you see something like the picture on the right. At $X = 2$ we have a sharp bent (indicated by the small dot). Such bents, peaks, and in general "unsmoothnesses" are candidates as points where the derivative isn't defined. But nonetheless the function is continuous at that point. This is a demonstration for the fact that if the derivative of a function exists at some point, then the function is continuous. But if the derivative doesn't exist at that point, then we can't say anything about the steadiness of the function at that point. It could be continuous, it could jump, anything is possible. Actually there are functions that are continuous everywhere and at the same time their derivative isn't defined anywhere at all!!! (Trabakoulas raises one eye brown and says "fascinating" ;-)) That means, dear math freaks, that these functions consist in a way… of infinite many infinitesimal small sharp bents. You wanna see one? Don't expect anything spectacular. trigonometric functions are already enough. For example the function:



$$w(x) = \sum_{n=1}^{\infty} a^n \cos\left(b^n \pi x\right)$$

with:

$$0 < a < 1 \;,\; b > 0 \;,\; \text{integer } b \;,\; \text{and } a\, b > 1 + \frac{3}{2}\pi$$

is such a beast. Don't even try to plot such a thing. You would need infinite resolution which no computing device on this world can give you. And if you still do, hopping to catch the real looking of the function by building the sum for the first, say 10 summands, you are going to get quite a surprise its time you zoom in. (OK, OK, I did that. I expanded:

$$\sum_{n=1}^{10} \left(\frac{1}{10}\right)^n \cos\left(60^n \pi x\right)$$

and plotted the resulting expression. Each time I zoomed in, I had new zig zags. And that with only the first 10 summands of the function.)

Btw, it was Weierstraß that discovered this function. And it was a quite fascinating discovery, an unexpected event, which happened because mathematics before him was practised the way it was practised. Let's take a look at math history to understand that. For quite a long time there was no clear and sharply defined definition for what "smooth function" and should be. Though mathematicians worked already with functions and derivatives, nobody knew how to sharply define what a point should be, where a derivative isn't defined. It was believed that a continuous function of one free variable is in general differentiable everywhere, except for some "pathological points", where the derivative doesn't exist. But then came the reconstruction of these terms, "continuous" and "differentiable", in terms of $\varepsilon - \delta$ (look at

SESELIMA). After this, it appeared that such unbelievable functions could be constructed, like the one above, that are nowhere differentiable and still continuous everywhere. And because such a function (still) looks like a marvel, and because mathematicians have often the flair for marvels, they searched and found these marvellous functions. All the magic of Copperfield is nothing compared to the Weierstraß function. An even bigger marvel however is, how much mathematics can be done, without these sharp definitions, simply by intuition. In the times before the invention of $\varepsilon-\delta$, there was still a huge amount of knowledge about functions. Admittedly, the real beauty was revealed after $\varepsilon-\delta$, but nonetheless it is still amazing how much was known before.

End of history, back to the present. The derivative of a function can be considered itself as a function. For example, enter e^{-X^2} and press DERVX to get $-\left(e^{-X^2}\ X\ 2\right)$. This function has itself a derivative which you can find if you press DERVX again: $\left(2\ X^2-1\right)\ e^{-X^2}\ 2.$ In algebraic syntax you could enter $\mathrm{DERVX}\left(\mathrm{DERVX}\left(e^{-X^2}\right)\right)$, or $\mathrm{DERIV}\left(\mathrm{DERIV}\left(e^{-X^2},X\right),X\right)$, or $\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(e^{-X^2}\right)\right)$, or any mixed form like $\frac{\partial}{\partial X}\left(\mathrm{DERIV}\left(e^{-X^2},X\right)\right)$. Expanding that will return the second derivative in one step. Higher derivatives can be found for undefined functions too. For example you can enter $F\left(X^2\right)$ and press DERVX twice to get the result $\left(2\ d1d1F\left(X^2\right)\ X^2+d1F\left(X^2\right)\right)\ 2$. The expression $d1d1F\left(X^2\right)$ denotes the derivative for the first variable of the derivative for the first variable of $F\left(X^2\right)$. It is a formal derivative of second order. Such higher order formal derivatives can be used exactly like the formal derivatives of first order (including user defined derivatives). And not only this. If you define the derivative of some particular order, the HP49G is able to find any higher order derivative using your definition. For example, if you enter $d1FUNC(X)=X\ e^{-X}$ and press DEF, then the variable $d1FUNC$ is created, which takes one variable from the stack and returns the first derivative of $FUNC(variable)$ for $variable$. If you now enter $d1d1FUNC(X)$ and expand, then the HP49G uses your definition to return the second derivative.

Another thing that I should tell here, is that such derivatives may also appear when the function that is differentiated is a built-in function. For example, if you take the derivative of $X!$ for X, the HP49G returns $d1!(X)$. This means the derivative of the function ! for the first variable. Here we see that the HP49G has also built-in functions of which it doesn't know how to take the derivative. Don't worry however, these functions are just a few and most of the time you can define a user defined derivative which will be used much like the derivatives that the HP49G knows how to take. Let's take for example the function GAMMA for which the HP49G returns $d1GAMMA(X)$ as its derivative. As you might already know:

$$\frac{\partial \ln\left(\Gamma(x)\right)}{\partial x}=\Psi(x)\ \Leftrightarrow\ \frac{1}{\Gamma(x)}\ \frac{\partial\Gamma(x)}{\partial x}=\Psi(x)\ \Leftrightarrow\ \frac{\partial\Gamma(x)}{\partial x}=\Gamma(x)\ \Psi(x)$$

Since the HP49G has the functions $\Gamma(x)$ and $\Psi(x)$ built-in ($GAMMA(x)$ and $Psi(x)$ respectively - unfortunately no greek letters Γ and Ψ), we can make the use defined derivative. Enter $d1GAMMA(X)=GAMMA(X)\ Psi(X)$ and press DEF to define the derivative. Let's try it. Enter $\mathrm{DERIV}\left(GAMMA(SIN(X)),X\right)$ and expand to get the result $COS(X)\ GAMMA\left(SIN(X)\right)\ Psi\left(SIN(X)\right)$ which doesn't contain any formal derivative anymore. If you enter $GAMMA\left(SIN(X)\right)$, then X, and then press DERIV, you get $COS(X)\ d1GAMMA\left(SIN(X)\right)$. This doesn't mean that our user defined derivative doesn't work in RPL syntax, you just have to press

EXPAND to convert the expression to COS(X) GAMMA(SIN(X)) Psi(SIN(X)). Since $\Gamma(x+1) = x!$ (this equation is an identity!), we obtain:

$$\frac{\partial \Gamma(x+1)}{\partial x} = \frac{\partial x!}{\partial x} \Leftrightarrow \frac{\partial x!}{\partial x} = \Gamma(x+1)\ \Psi(x+1)$$

which we can use to make a user defined derivative for the built-in function !. We only need to define d1(X) = GAMMA(X+1) Psi(X+1). (Alternatively if we already have defined d1GAMMA(X) we can also define d1(X) = d1GAMMA(X+1)). But the problem is that we can't enter the name d1 because the HP49G things that we mean the factorial of d1. So we have to search in the trick box again. As we have seen, the expression d1(X) is returned when we try to take the derivative of !. We can use this answer to built up our definition - somehow reminds me of "anything that you say can be used against you ;-). Enter X!, then X, and press DERIV to get d1(X). Now, enter the expression GAMMA(X+1) Psi(X+1) and press = to get the result d1(X) = GAMMA(X+1) Psi(X+1). Press DEF to make the definition. Let's try that. Enter:

$$\frac{\partial}{\partial X}(X!)$$

and expand. Oops, it stayed at d1(X). Perhaps another expand? Doesn't help either. The same happens if you start with DERIV(X!,X) or if you use RPL syntax. The resulting function d1 in the algebraic object d1(X) doesn't seem to want to be evaluated. You can evaluate it, expand it, do anything you want. The darn thing is there, the HP49G is able to evaluate any user defined function that exists in the current path, but this damned d1 just doesn't want to get evaluated though the internal structure of the expression d1(X) is the same like the structure of F(X), G(X), or even VPN(X). For me this is one of the most mysterious questions about the HP49G. **Why doesn't d1(X) get evaluated when the user defined derivative exists?** But we don't give up yet. Let's try to see if there is any difference between the d1 that exists as a variable in the current directory and the d1 that the derivation of X! returns. Press VARS (second page of menu PRG/MEM/DIR) to get a list of the variables in the current directory. If d1 was the last thing that you defined the list will look like {d1 other names}. If the list looks different, then find out at which position d1 is. Enter the number of the position of d1 in the list and press GET to extract d1 from the list. (We do all this because we can't directly enter the name d1. If we do so the built-in command line parser will thing that we mean the factorial of the variable d1, and not the derivative of the function ! for the first variable.) Now, enter X!, then X, and then press DERIV to get d1(X). We have to extract the name d1 out of the algebraic object. Press →LST (second page of menu 256) to get the list {X 'd1' #1h }. The object d1 is at position 2, so enter 2 and press GET. Do you see the difference? The object in stack level 2 is the name 'd1', while the object on stack level 1 is the *algebraic object* d1 that consists only of the name d1. Enter X, and press OVER to get a copy of d1 on stack level 1. Press EVAL. Evaluation of the algebraic object d1 just puts the name 'd1' on the stack. You have to evaluate once again to get GAMMA(X+1) Psi(X+1). This is the normal way of evaluation of an algebraic object containing only one name of an existing user defined function. And here is the question: Why isn't then the *expression* d1(X) evaluated to GAMMA(X+1) Psi(X+1) even after two evaluations? If you had pressed EXPAND instead of EVAL, then the first EXPAND would turn the algebraic object d1 to the name 'd1' and subsequent expansions would leave the name d1 unchanged. Anyway, the user defined derivative d1 is useless in this form. We must find another way. But before we do that, let's allow my alter ego HULK come out and speak for a moment:

…thank you very much indeed, ACO, that you put so much power in the HP49G, yet made it also so full of mysteries that its usage requires such headaches. And for me, the user, explanations about inner workings of the function !, or any other command, just don't count. I don't want to know how the HP49G does its work, I want a CAS that is uniform in its behaviour on the interface between human and machine… grrrrr!

OK Hulk, de-green again :-)

Now we have to purge d1. Since direct input of d1 wouldn't work, you have to use again VARS, and extract the name d1 from the list of variables. With the name d1 on stack level 1 press PURGE to get rid of that beast.

Thanks goodness there is the synonym FACT for the function !. So perhaps we use that? Hmm, let's try. Enter d1FACT(X) = GAMMA(X+1) Psi(X+1) and press DEF. Now, go to the EQW and enter:

$$\frac{\partial}{\partial X}(\mathrm{FACT}(X+1))$$

Put that on the stack and expand to get GAMMA(X+1+1) Psi(X+1+1). It works! But we have a new problem. We can't use that with RPL syntax. If you enter X+1 and then FACT, the HP49G returns X!(X+1). (Automatic simplification using the rule (X+1)!= X!(X+1). If you enter X and then FACT, then the result is X!. All occurrences of FACT are replaced by !. If you now try to take the derivative for X you will of course end up with expressions containing d1, which we don't want to have. But hey! We could make a program that converts all occurrences of ! to FACT. Enter the program:

```
<<
  { '&A!' 'FACT(&A)' }
  ↑MATCH DROP
>>
```

and store the program in → FACT. If you work with RPL syntax, you can convert all occurrences of ! to FACT before derivation, in order to get an expression that contains d1FACT. For example, enter again (X+1)!. Expand that to get X!(X+1). Now, press →FACT to convert that to FACT(X)(X+1). Enter X and press DERIV to get d1FACT(X)(X+1)+FACT(X). Expand this to get (X+1) GAMMA(X+1) Psi(X+1)+X!.

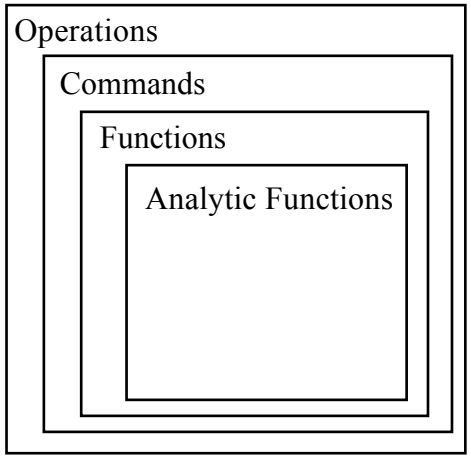
Alternatively you can enter (X+1)! and press →FACT without expanding. The result is FACT(X+1). Now you can enter X and press DERIV to get d1FACT(X+1). Expanding this you will get GAMMA(X+1+1) Psi(X+1+1). If you want you could also make a program that matches GAMMA(something) to (something−1)!, in order to get results that contain only !, but no GAMMA. For example the code

```
<<
  { 'GAMMA(&A!)' '(&A-1)!' }
  ↑MATCH DROP
>>
```

would do this conversion.

Perhaps now that we have seen that there are built-in functions for which the HP49G doesn't know how to take their derivatives out of the box, it is a good time to say some words about operations, commands functions and so on. Anything that the HP49G can perform is an operation. When you press + to add two numbers, the HP49G carries out an operation. When you press VAR to go to variables menu, the HP49G carries out another operation. Of all operations, those that are

programmable are called commands. So for example, + is a command, because it is programmable. But pressing the key [VAR] isn't programmable and so it isn't a command. There are of course programmable equivalents for many operations that are not programmable. For example you can use 2 MENU to get the variables menu. On the HP49G when no other programmable equivalent exists, there is always the possibility to use KEYEVAL, to simulate a key press. For example, pressing [VAR] can be also programmed with 31 KEYEVAL. We see until now that the commands are a subset of all available operations. Commands that are allowed in algebraic objects are called functions. For example + is a function because it is allowed in an algebraic object, but the command PATH is not a function because you can't put that in an algebraic object. Again, functions are a subset of commands. And last we have those functions for which the HP49G provides an inverse and a derivative. These functions are analytic functions in the terminology of the HP49G. They are a subset of the functions. So we have an architecture like the picture on the right.

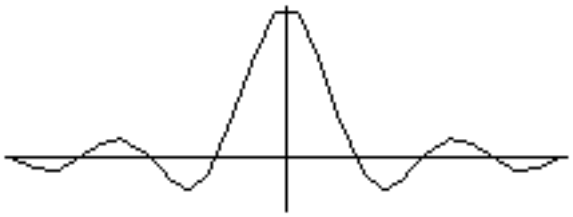


The sentence "provides an inverse and a derivative for a function" means that the HP49G can solve function(x) = y analytically for x, and can find the derivative $\frac{\partial}{\partial x}(\text{function}(x))$ without any user intervention. In the terminology of the HP49G for example, the functions, ! or GAMMA, are not analytic, because the HP49G can't solve GAMMA(X) = Y analytically (though it can do that numerically for numeric values of Y), and can't find $\frac{\partial}{\partial X}(\text{GAMMA}(X))$ (though we can make a user defined derivative). Note that the meanings of "function" and "analytic function" for the HP49G are not 100% identical with their mathematical meanings.

The HP49G provides also powerful tools in the plotting environment for finding slopes and derivatives in an interactive way. Let's try one example. Go to the PLOT SETUP screen and choose plot type Function. Enter:

$$\frac{\text{SIN}(X)}{X}$$

as the function to plot, and X as the independent variable. After this, go to the PLOT WINDOW − FUNCTION screen and set H−View from −12.5 to 12.5. Press ERASE and AUTO (for automatic scaling of V−View). When the HP49G finishes automatic scaling, press DRAW to let the HP49G plot the function. Now we are going to add the derivative of the function to the already existing plot. We don't need to move out of the plotting environment, find the derivative, add it to the functions to plot, and redraw. Press FCN which brings up a menu with many tools that relate to plotting functions. Move the graphics cursor some 10 pixels to the right and press SLOPE. This operation finds and displays the slope of the function at the current horizontal coordinate. It also puts a copy of the slope on the stack. Press any menu key to display the menu again, and press [NXT] to go to the second page of the menu. Now, press F'. This finds the derivative of the function, adds it to the functions to plot, and redraws both functions. It also leaves the menu FCN. Now the reserved variable EQ contains the list:

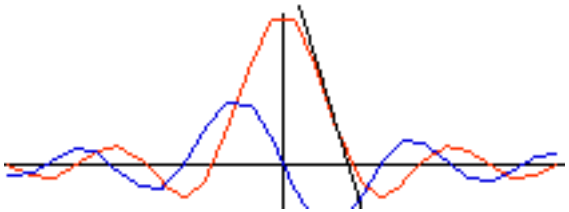


$$\begin{bmatrix} X & \frac{X\ \cos(X)-\sin(X)}{X^2} & \frac{\sin(X)}{X} \end{bmatrix}$$

The HP49G added the derivative to the functions to be plotted. Press FCN again. Since we have two functions in the list, the question is, on which of them will the tools of the menu FCN operate? The answer is: On the first. That means, if you for example move the cursor somewhere and press SLOPE, the slope will be found for

$$\frac{X\ \cos(X)-\sin(X)}{X^2}$$

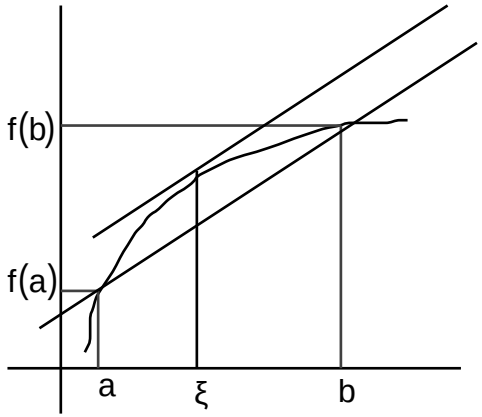
that is for the derivative that we found. You can take a look at the function on which the tools operate by pressing ◄ and then ▼. This displays the first function in the list for about one second on the top of the screen. If you want to switch to the next function in the list, move the cursor a bit to the right or to the left, press NXT and then NXEQ. The cursor moves then to the next function which gets displayed on the bottom of the screen. (We move the cursor a bit away from $X=0$, because the HP49G would use $\frac{\sin(X)}{X}$ to find the vertical coordinate. Since at $X=0$ we have division by 0, this causes the HP49G to leave the plotting environment because of error.) Pressing NXEQ also rolls the items in the equation list. The first equation is moved to the last place. The second moves to the first place. The third to the second, and so on. Now press some menu key again to display the menu, move the cursor again some 10 pixels to the right and press TANL, to draw the tangent line of $\frac{\sin(X)}{X}$ at that point and display its equation on the bottom of the screen. A copy of this equation is placed on the stack for later use. If you leave the plotting environment and return to the stack you will see the slope and the equation of the tangential line that the HP49G put there for you. (The equation of the tangential line isn't added in the list of equations to plot.)



Let's move on now to the other calculus stuff. First of all, automatic scaling has created the variable X. You may want to purge it now, as it is often the variable VX and this interferes with the CAS of the HP49G. One of the calculus statements, that doesn't sound very interesting at first, but nonetheless has immense consequences, is:

If a function $y=f(x)$ is continuous in $a\le x\le b$ and differentiable in $a<x<b$, then there exists always some value ξ between a and b, such that $\frac{f(b)-f(a)}{b-a}=\left.\frac{\partial f(x)}{\partial x}\right|_{x=\xi}$.

That means geometrically that the slope of the tangent line of $f(x)$ at $x=\xi$ is equal to the slope of the secant that goes though the points $(a,f(a))$ and $(b,f(b))$.



If we have a function $f(x)$, two points a and b, can we then find on the HP49G what the equation of the secant line is, that goes through the points $(a,f(a))$ and $(b,f(b))$? Yes, we can. This operation wasn't included in the tools of the menu FCN of the plotting environment, but nonetheless it is easy to do that. Suppose that we have the function $F(X)=(X^2-1)\ e^{-(X^2)}$ and we want to find the equation of the line that goes through $(0,F(0))$ and $(1,F(1))$. Enter:

$F(X) = \left(X^2 - 1\right)\ e^{-\left(X^2\right)}$

and press DEF to create the user defined function F. We are going to need the user defined function F and the commands R → C and DROITE. Since they reside all in different menus, we make a new temporary menu that contains them all. Enter the list {F R → C DROITE} and press TMENU. This creates a temporary menu with the commands that we need. The menu exists only until we leave it. Enter 0, the first X-coordinate. Press ENTER to make a copy of it on stack level 2, and press the menu key F. This returns −1, the Y-coordinate that corresponds to X = 0. Press the menu key R→C to create the complex number (point on the plane) (0.,−1.) out of the numbers on stack levels 1 and 2. Now, enter 1, press ENTER to make a copy at stack level 2, press F and then again R→C to get the second point, (1.,0.). Now, press DROITE. The command DROITE takes two points from the stack and returns the equation of the line that goes through these points. The points can be two complex numbers, in numeric or in algebraic style. That means that we can use for example (0.,−1.) and (1.,0.) (like we did), or 0 − i and 1. The equation that DROITE returned is Y = X − 0. − 1.. It contains numeric (real) values because we used numeric complex numbers. Press XQ to turn them to exact integers and EXPAND to get Y = X − 1, the equation of the line that goes through (0,−1) and (1,0). All the above is of course easy to program.

```
<<
  PUSH                    @Save user's settings
   → f v a b              @Store in local variables
  <<
    a b 2 →LIST           @Make a list of the two coords
    1 << →NUM >>          @Turn them to numbers
    DOSUBS
    f
    v PICK3 =             @make list {var=a var=b} and
    SUBST                 @substitute in function
    EXPAND R→C            @Make list {(a,f(a) (b,f(b)}
    OBJ→ DROP             @Explode it
    DROITE                @Find secant line
    EXPAND
    RCLVX v = SUBST       @Use variable of function
  >>
  POP                     @Restore user's settings
>>
```

Store the program in SECLINE. If you want, you can add XQ after the last EXPAND. The program takes from the stack the function, its variable, and the two x-coordinates. Let's test it. Enter $SIN(X)\ e^{-X}$, X, $\frac{\pi}{3}$ and $\frac{3}{4}\pi$, and press SECLINE. The result is:

$Y = -\left(.180967365884\,X - .493414050108\right)$. Note that this program will work only for points that are evaluable to numbers, because the command DROITE doesn't work with symbolic arguments. If we want a program that does its work with any possible argument type, we unfortunately have to abandon the easiness of DROITE and do all work ourselves. But we can get support from the already existing program ΔQUOT. This program doesn't expect two x-coordinates but rather one x-coordinate and Δx. So, if we want to use it for a secant, we have to transform the two x-coordinates. This is easily done by not providing the x-coordinates a and b themselves, but a and b − a (instead of Δx). The program then returns the slope of the secant line. The only thing that remains then is to program the calculation of the constant of the equation of the secant line. If the function that we have is f, and the x-coordinates are a and b respectively, then this constant is given by:

$$\frac{f(a)\ b - f(b)\ a}{b - a}$$

The program SECLINE that comes with this document uses this method.

```
<<
  PUSH                    @Save user's settings
   → f v a b              @Store in local variables
  <<
    'Y'                   @Enter Y
    f v a = SUBST b *     @Find (f(a)*b-f(b)*a)/(b-a)
    f v b = SUBST a *
    - b a - /
    f v a b a -           @Create arguments for ΔQUOT
    ΔQUOT                 @and call ΔQUOT
    v * + =               @Built up secant line equation
  >>
  POP                     @Restore user's settings
>>
```

It has the disadvantage that we do all work ourselves. But it is more flexible. Let's do an example. Enter e^X, X, 0 and 1. Press SECLINE. The result is $Y = X\ e^1 - (X-1)$, the equation of the secant line that goes through the points $(0, e^0)$ and $(1, e^1)$.

Now that we have that, you might suspect what comes next. Make a program that finds the equation of the tangent line of some function at a given point x_0. We can find the slope of the line easily by finding the derivative and substituting $x = x_0$. That means, we have to find:

$$\left.\frac{\partial f(x)}{\partial x}\right|_{x=x_0}$$

The constant of the equation of the tangent line is given by:

$$f(x_0) - \left.\frac{\partial f(x)}{\partial x}\right|_{x=x_0} x_0$$

which means that we can use the quantity

$$\left.\frac{\partial f(x)}{\partial x}\right|_{x=x_0}$$

again. Here is the listing of the program TANLINE that takes a function f, its variable x, and a coordinate x_0 from the stack, and returns the equation of the tangent line at the point $(x_0, f(x_0))$.

```
<<
  PUSH                    @Save user's settings
   → f v a                @Store in local variables
  <<
    'Y'                   @Enter Y
    f v ∂                 @Find ∂f/∂v
    v a = SUBST           @Substitute v=a
    DUP v *               @Find ∂f/∂v|v=a
    f v a = SUBST         @Find f(a)
    ROT a * -             @Find f(a)-a ∂f/∂v|v=a
    + EXPAND =            @Built up tangent line equation
  >>
  POP                     @Restore user's settings
>>
```

We test the program. Enter SIN(X), X and then 0. Press TANLINE to get $Y = X$, the tangent line equation of SIN(X) at (0,SIN(0)).

Another interesting problem. Assume that you have a secant line that goes through two points of some function. Then you search for a point of the function, whose tangent line is parallel to the secant line. We can make a program that takes a function f, its variable x, and two coordinates a and b from the stack, finds the equation of the secant line, and then finds the point whose tangent line is parallel to the secant line. This problem is also easy to solve. In order for the two lines to be parallel, they must have the same slope. The slope of the secant line is given by:

$$\frac{f(b)-f(a)}{b-a}$$

The slope of the tangent line is given by the expression:

$$\left.\frac{\partial f(x)}{\partial x}\right|_{x=x_0}$$

We have to find for which x_0 the equation

$$\frac{f(b)-f(a)}{b-a}=\left.\frac{\partial f(x)}{\partial x}\right|_{x=x_0}$$

is satisfied. On the HP49G it suffices to solve the equation

$$\frac{f(b)-f(a)}{b-a}=\frac{\partial f(x)}{\partial x}$$

for x. Again we can use ΔQUOT for finding the quantity

$$\frac{f(b)-f(a)}{b-a}$$

Note however that in most cases the equation will not be analytically solvable and so we will use also numerical solving. Here is the listing of the program TANPARSEC (TANgent line PARallel to SECant line - do you have a better name?)

```
<<
  PUSH                    @Save user's settings
   → f v a b              @Store in local variables
  <<
    f v ∂                 @Find ∂f/∂v
    f v a b a - EXPAND     @Find (f(b)-f(a))/(b-a)
    ΔQUOT
    DUP "Slope" → TAG     @Label result
    UNROT = v             @Create arguments for ZEROS
    "Try analytically"    @Inform user what's going on.
    1 DISP
    IFERR                 @If trying to solve errors
      ZEROS
    THEN                  @then
      "Failed.
Try numerically"          @Inform user
      1 DISP
      b a - 2. / ROOT     @and try numerically
      v PURGE             @Purge created variable
    END
  >>
  POP                     @Restore user's settings
>>
```

Let's try the program. Enter SIN(X), X, $\frac{\pi}{2}$ and $\frac{3\ \pi}{2}$. Press SECLINE to get:

$$Y=\frac{2\ \pi-2\ X}{\pi}$$

This is the equation of the tangent line. Now, re-enter SIN(X), X, $\frac{\pi}{2}$ and $\frac{3\ \pi}{2}$, and press TANPARSEC. The results are the tangent objects:

Slope: $\frac{-2}{\pi}$

and

X: $\left\{\left(-\frac{\pi}{2}+2\ n1\ \pi+\text{ACOS}\left(\frac{-2}{\pi}\right)\right)\ \left(\frac{\pi}{2}+2\ n1\ \pi+\text{ACOS}\left(\frac{-2}{\pi}\right)\right)\right\}$

That means that the slope of the tangent (and the secant) line is $\frac{-2}{\pi}$, and that the tangent line at the points given in the list has this slope. We get more than one solutions because the sine is a periodic function. Now, we want the equation of the tangent line at those points. We work with the second solution:

$$2\ n1\ \pi + \text{ACOS}\left(\frac{-2}{\pi}\right)$$

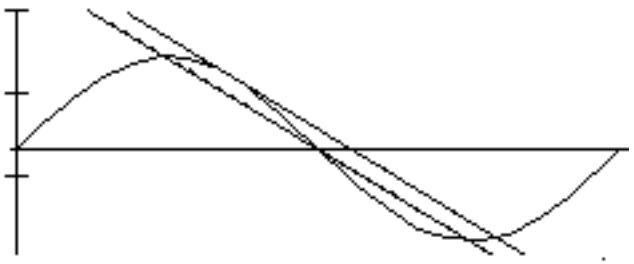
Enter 2 and press GET to extract it from the list. We will work with n1= 0, so we must substitute this value in the solution. Enter n1= 0 and press SUBST, then EXPAND to get:

$$\text{ACOS}\left(\frac{-2}{\pi}\right)$$

This is the point where we want to find the tangent of SIN(X). Enter SIN(X), then X, and then press ROT to put the arguments in the right order. Press TANLINE to get:

$$Y = \frac{2\ \text{ACOS}\left(\frac{-2}{\pi}\right) - 2\ X + \sqrt{\pi^2 - 4}}{\pi}$$

This is the equation of the tangent line. Let's plot the function, the secant and the tangent line together. Press EQ→ and NIP to extract the right hand side of the equation of the tangent line. Press NIP again to get rid of the slope. Press ▶ to bring the equation of the secant line on stack level 1 and EQ→ and NIP to extract the right hand side of the equation of the secant line. Enter SIN(X), then 3, and then press →LIST and press STEQ to store the list in EQ. Set plot type Function, independent variable X, horizontal view range from 0 to 6.28, and vertical view range from −1.2 to 1.5. Now DRAW to get a plot that looks like the picture on the right. Of course all this can also be wrapped in a program that does the all work automatically.



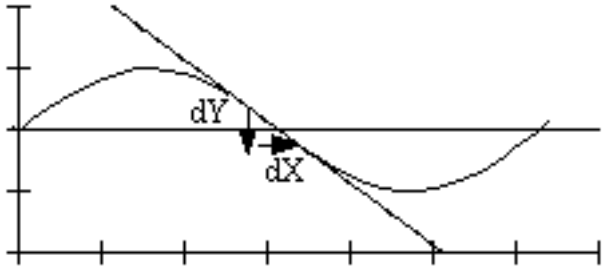
Another question that one might ask is, how can we represent differentials of functions on the HP49G? Is there any way? For example, suppose that we have the function $Y = X^2$. Its differential dY is defined as:

$$dY = \frac{\partial X^2}{\partial X}\ dX \Leftrightarrow dY = 2\ X\ dX$$

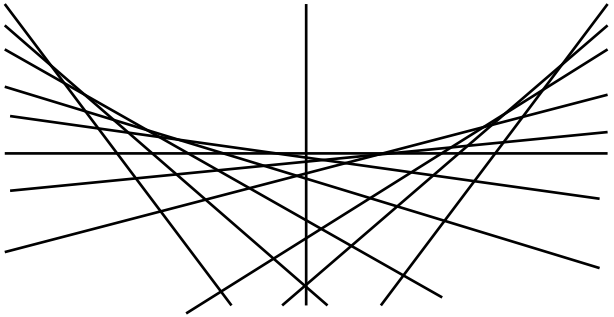
Of course we can take the derivative of the function, but how can we denote the differential dX of the independent variable? One way would be to simply append the small "d" in front of the name of the variable. The question is how much work can be done with such an expression. The answer is, at least it is the possible to store some value in variable dX (or in general dvariableNameϵ), and evaluate to get the local description of the function at some given point. Consider for example the program:

```
<<
  → f v                  @Store in local variables
  <<
    f v ∂                @Find ∂f/∂v
    "'d" v + "'" +       @Make string "'dv'"
    OBJ→                 @Turn string to name
    *
  >>
>>
```

Store this in DY. Enter SIN(X), X and then press DY to get COS(X) dX. We see that the function SIN(X) can be described locally, i.e. in the neighbourhood of some given point X_0 as COS(X_0) dX, where dX is some tiny quantity. When X has a change of dX, then SIN(X) has a change of COS(X_0) dX. For example let's see what SIN(X) does at X = π. Enter X = π and press SUBST and EXPAND to get −dX. This means that at around X = π, when X has a variation of dX, then SIN(X) has a variation of −dX. As you can see, the local description of the curve can be approximated using differentials. They say how much the function varies at a given point, when the independent variable varies a tiny little dX. Geometrically this is like wanting to represent the function by many tangent lines, one at every point. A good way to represent that is to draw them without drawing the function itself. For example let's draw the tangent lines of X^2 from −2 to 2, in steps of 0.2. We will use the program TANLINE to find all the equations of the tangent lines. Enter `<<'X^2' 'X' a TANLINE EQ → NIP>>`, 'a', −2, 2 and .2. Press SEQ to evaluate (run) the program for all values of a from −2 to 2 in steps of 0.2. It takes some seconds to run, so be patient. When it finishes, stack level 1 contains a list with the equations of the tangent lines. Press STEQ to store the list in EQ. Set plot type Function, independent variable X, horizontal view range from −2 to 2, and vertical view from −1 to 4. Now press ERASE and then DRAW to see the plot. It takes quite a long time to start plotting, and even longer to finish the plot, so be patient. When it finishes you have the impression that the parabola was drawn, though we plotted only straight lines.



The higher differentials can be calculated by using the code in DY repeatedly. If the variable dX which is introduced by DY is not the variable of derivation and is not in the original function, it will not interfere with our operations.



```
<<
  → f v n                 @Store in local variables
  <<
    f                     @Put f on stack
    1 n START             @Do n times
      v ∂                 @Find ∂f/∂v
      "'d" v + "'" +      @Make string "'dv'"
      OBJ→                @Turn string to name
      *
    NEXT
    FACTOR                @Collect all dv
  >>
>>
```

Store the new code in DY and let's test it. We will find the second differential of SIN(X) e^{-X}. Enter SIN(X) e^{-X}, then X and then 2. Press DY to get $-(2\ e^{-X}\ \mathrm{COS}(X)\ dX^2)$. As you can see the differential of the independent variable isn't at the very end of the algebraic. But this is only a cosmetic problem.

The last thing at which we take a look here is a small comparison between the notions $\frac{\partial}{\partial X}(F(X))$ and d1F(X) that both are possible on the HP49G. The algebraic (no pretty print) form of the notion $\frac{\partial}{\partial X}(F(X))$ is

$\partial X(F(X))$. It looks quite similar to d1F(X). But the differences are huge. In the notion $\partial X(F(X))$ the sequence of characters ∂X is not a name. The second character is the variable of differentiation and is the *second argument* of the function ∂. The HP49G displays $\partial X(F(X))$ but internally it means $\partial(F(X),X)$. It is only a special display and nothing more. Quite different in d1F(X), the sequence of character d1F is a *name*. It is a normal name that you can type in, exactly like any other name. There is no built-in function d1F or d2VPN or d3RCOBO. So we can define our own functions, that is, we make user defined derivatives. When the CAS of the HP49G has to manipulate some expression, it apparently checks if names are present, that are constructed in the way dNumberName. If it finds any, then it uses its built-in knowledge about derivatives. So for example it is possible to enter X(X), then X, and then press DERIV to get d1X(X). This is a quite pathological example, and I use it only for explanations, though it might as well have its usefulness. (He, he, you guessed right, in some of the next parts we will perhaps use it... perhaps! ;-)) In this example the dependence of the function X on itself is somehow "hidden" in the derivative. The function name has become d1X and its argument is X, i.e. we have two different names out of two identical names of the expression X(X). There is yet another difference, which I think will can be understood better, if we focus on the difference between variables and names on the HP49G. A name is a single object (object type 6.), like for example X, Y, VOLUME, or even Karagiaouroglou. (OK, the last example might be no name at all ;-)) But a variable has become quite an abstract concept on the HP49G. On the HP48 the world was easier, variables were names, end of story. But on the HP49G a variable can be sometimes a name, sometimes an expression. A variable is something that can… vary, no matter if it is a name or an expression. What the CAS considers a variable is a *rational variable* and not necessarily a simple name. For example, if you enter SIN(X Y)−COS(Z), and press LNAME, then the HP49G returns the vector with the names that appear in the expression, [X Y Z]. But if you press LVAR, the HP49G returns the vector of rational variables, [SIN(X Y) COS(Z)]. Apparently the two quantities SIN(X Y) and COS(Z) are for the HP49G two entities, two *things*, that are treated as wholes in many (all?) symbolic manipulations. And here we have a good starting point for understanding the difference between the two notions $\frac{\partial}{\partial X}(F(X))$ and d1F(X). (At this point it would be good to purge F if you still have it from the previous pages.) The first notion, $\frac{\partial}{\partial X}(F(X))$, is the derivative for the *name* X. The second, d1F(X), is the derivative for the *rational variable* X. Of course in this example both are the same, but if we write X^2+X instead of X between the parentheses, then we get:

$$\frac{\partial}{\partial X}\left(F\left(X^2+X\right)\right)$$

and:

$$d1F\left(X^2+X\right)$$

Now we see that

$$\frac{\partial}{\partial X}\left(F\left(X^2+X\right)\right)$$

still means the derivative for the *name* X. If you expand it you get $(2\ X+1)\ d1F\left(X^2+X\right)$. On the other hand if there is no definition of the derivative d1F(X), then expanding $d1F\left(X^2+X\right)$ returns $d1F\left(X^2+X\right)$ unchanged, because this is the first derivative of F for whole *rational variable* X^2+X. Note that the second notion counts arguments

(variables) simply by position. The construction dNumberFunction(var1,var2, ,varN) separates the arguments by commas. Between the commas we have slots where the rational variables go. So, entering

$$\frac{\partial}{\partial X}\left(F\left(\frac{X}{Y},\frac{Y}{X}\right)\right)$$

and expanding, returns

$$\frac{-Y}{SQ(X)}\ d2F\left(\frac{X}{Y},\frac{Y}{X}\right)+\frac{Y}{SQ(X)}\ d1F\left(\frac{X}{Y},\frac{Y}{X}\right)$$

In these expressions, though we started with a derivative for a name, we end up with derivatives for a rational variable. This doesn't mean that the result is incorrect. Quite the contrary it is correct and if it is used wisely it is also very useful.

Note also that the HP49G allows entering for example d100F(X). Expanding this doesn't do anything, though the function F has only one and not 100 variables. This means that we have a way to enter derivatives of non-existing variables which will not disappear when we expand or do something else. With the notation $\frac{\partial}{\partial X}(F(X))$ we can't do that. If you expand for example $\frac{\partial}{\partial Y}(F(X))$ (i.e. derivation for a non existing variable), then you get 0. In the above comparison one could as well use DERIV(F(X),X) instead of $\frac{\partial}{\partial X}(F(X))$.

We came to the end of the first part of this marathon. If the above things were not very exciting, then wait until the next part, which will definitely come without delays. What comes in the next part was originally thought for this part. But then there would be too much important stuff about the "technical" part of the commands for derivation, which wouldn't have been covered at all. So I decided to reorganise things (once again) and do first all that stuff that you should know before we proceed. I hope you enjoyed it.

Greetings,
Nick.

Basic Calculus with the HP49G - Volume 1 - Part 2

Hi again!

In the first part of the Basic Calculus Marathon, we spent a lot of time by examining the "technical" part of the derivation commands and their behaviour in many different cases of syntax and arguments. We continue this marathon examining what rules of derivation are known to the HP49G, and what we can program ourselves.

The HP49G is able to perform (almost) all rules of differentiation. Most of the time you don't even notice that, but the HP49G finds derivatives following those rules. We are going to take a closer look to these processes by activate the step by step feature. When this feature is activated then differentiations can be followed from one step to the next.

Before we go further, purge the variables and user defined functions that we used in the first part, because they may interfere with what we are going to do now. Enter {R S V W Z Q Z1 Z2} and press PURGE to get rid of them. If the variables don't exist, then PURGE will not error out. It will simply remove the list from the stack. (PURGE behaves in the sense "Delete file if it exists".)

The first derivation rule that we are going to examine is:

$$\frac{\partial}{\partial X}(C\ F(X)) = C\ \frac{\partial}{\partial X}(C\ F(X))$$

Does the HP49G know that? Let's see. Enter

$$\frac{\partial}{\partial X}(C\ F(X))$$

and press EVAL to get

$$C\ \frac{\partial}{\partial X}(F(X))$$

If you press EVAL again, then you get C d1F(X). Notice that the step by step feature works only for algebraic entry of the whole derivative and evaluation. If you enter C F(X), then X, and then press ∂, you will get the end result, without the steps in-between. You have to enter your derivative in the form

$$\frac{\partial}{\partial \text{name}}(\text{expression})$$

and press EVAL. Any other way will give you the end result in one step. Let's have a particular example, enter

$$\frac{\partial}{\partial Y}(A\ Y^2)$$

and evaluate to get

$$A\ \frac{\partial}{\partial Y}(Y^2)$$

Evaluate again to get 2 Y A. Of course you can also expand

$$\frac{\partial}{\partial Y}(A\ Y^2)$$

to get A 2 Y immediately without any steps in-between, but we examine here if and how the HP49G applies derivation rules. The opposite direction, collecting the constant in front of the derivative and multiply it with the function inside the derivative, is not possible on the HP49G out of the box. But we can (of course ;-)) program that. We can use the command ↑ MATCH to convert any occurrence of the pattern

$$\text{constant}\ \frac{\partial}{\partial \text{name}}(\text{function})$$

to the pattern

$$\frac{\partial}{\partial \text{name}}(\text{constant}\ \ \text{function})$$

But we must watch out here. We can't blindly do this matching. We must check first if constant is itself a function of name. If it is, then we can't put it inside the parentheses of derivation. We are allowed to do that only if constant doesn't depend on name. The command ↑ MATCH offers the possibility to check some condition and do the matching only if the condition is true. The condition that we must use is in general "variable not in constant". We have the command LNAME, which returns a vector of all names in an algebraic object, or an empty list if the algebraic object doesn't contain any names. We can check if name is in that vector and decide what to do accordingly. But… to do this we have to give the command ↑ MATCH a list containing three algebraic objects, the third of which is the condition. And LNAME is a command, not a function, and thus it can't be put in an algebraic object. What to do in such a case? We have to somehow smuggle what LNAME does in an algebraic object. Here comes the flexibility of the HP49G. Imagine some program that does anything it does, but returns a single object which is allowed in an algebraic. For example, suppose that you have the program

```
<<
    π  → NUM
    1 10 START
      COS
    NEXT
    XQ
>>
```

stored in MULTPI10. If you enter the expression $\text{MULTPI10}^2\ X$ and expand, then the program will be evaluated and its result,

$$\frac{36293}{49621}$$

will replace the variable MULTPI10 in the expression, giving:

$$\left(\frac{36293}{49621}\right)^2 X.$$

The expansion then proceeds and returns the result

$$\frac{31317181849\ X}{2462243641}$$

Since the result of MULTPI10 something allowed in algebraic objects, everything works OK. Another example: Suppose that you have the program

```
<<
   → x
   <<
     x TVARS SIZE
   >>
>>
```

stored in NUMTVARS. The program takes a number as argument and returns all variables in the current directory, that are of the same type like the number. If you enter the algebraic object NUMTVARS(8) +10 and expand, then the HP49G will return the sum of the number of programs (type 8.) in the current directory and 2.

In exactly the same way we write the program

```
<<
   SWAP LNAME NIP        @Return list of names in alg.
   IF DUP TYPE 29. ==    @If result is symbolic vector
   THEN                  @then
     AXL                 @convert it to a list
   END
   SWAP POS              @Position of name in list.
>>
```

and store it in POSNAME. The program takes two arguments. An algebraic (or name) on stack level two, and a name on stack level 1. It returns the position of the name in the vector of variables of the algebraic object. If you enter X^2-Y+Z and Z, then pressing POSNAME returns 2. because Y is in the second position of the vector of names of X^2-Y+Z. If you enter X^2-Y+Z and A, then pressing POSNAME returns 0., because A isn't contained in X^2-Y+Z. The important thing is that you can also do the same in algebraic syntax. You can enter $\text{POSNAME}(X^2-Y+Z,Y)$ and EXPAND. This will also set approximate mode on, switch back if you don't want it. Now we can use POSNAME(expression,name) in any algebraic object. We can use it as the testing condition for the command ↑ MATCH. Let's do an example first. Enter

$$C\ \frac{\partial}{\partial X}(F(X))$$

Now enter the list:

$$\left\{ \&c\ \frac{\partial}{\partial \&v}(\&f)\quad \frac{\partial}{\partial \&v}(\&c\ \&f)\quad \text{NOT POSNAME}(\&c,\&v) \right\}$$

Press ↑MATCH. The result is

$$\frac{\partial}{\partial X}(C\ F(X))$$

on stack level 2, and 1. on stack level 1, which shows that pattern matching was performed. The HP49G evaluated NOT POSNAME(&c,&v), the testing condition. Since variable &v (i.e. X) was not in expression &c (i.e. C), the program POSNAME returned a 0. This was negated and the result of the testing condition was 1., which stands for true. The testing condition was evaluated to true and so pattern matching was performed. Enter

$$X\ \frac{\partial}{\partial X}(F(X))$$

then press CMD to get the popup with the last 4 commands. Select the list with the pattern matching arguments and press ENTER to put it in the command line. Press ENTER again to put it on the stack. Now, press ↑MATCH again. The result now is

$$X\ \frac{\partial}{\partial X}(F(X))$$

on stack level 2, and 0. on stack level 1, which shows that pattern matching was not performed because &v (i.e. X) was in expression &c (i.e. X). We will use this and other similar tricks later on, to make a program that collects differential forms, something that the HP49G can't do out of the box.

Next rule that we examine is the derivation rule for sums, which is:

$$\frac{\partial}{\partial X}(F(X)+G(X))=\frac{\partial}{\partial X}(F(X))+\frac{\partial}{\partial X}(G(X))$$

Enter

$$\frac{\partial}{\partial X}(F(X)+G(X))$$

and evaluate. The HP49G returns

$$\frac{\partial}{\partial X}(F(X))+\frac{\partial}{\partial X}(G(X))$$

If you evaluate again then you get $d1F(X)+d1G(X)$. We also do a particular example. Enter

$$\frac{\partial}{\partial X}\left(X^2\ A-X\ 2\right)$$

and evaluate. The result is

$$\frac{\partial}{\partial X}\left(X^2\ A\right)-\frac{\partial}{\partial X}\left(X\ 2\right)$$

which shows that the HP49G used the derivation rule for sums. Evaluate again to get

$$A\ \frac{\partial}{\partial X}\left(X^2\right)-2$$

Here the HP49G used the rule of derivation of an expression multiplied by some constant. Evaluation of the term

$$\frac{\partial}{\partial X}\left(X^2\ A\right)$$

resulted in

$$A\ \frac{\partial}{\partial X}\left(X^2\right)$$

which still contains a derivative. Evaluation of the term

$$\frac{\partial}{\partial X}\left(X\ 2\right)$$

resulted in

$$2\ \frac{\partial}{\partial X}\left(X\right)$$

which was directly simplified to $2\ 1=2$. Evaluating again we get $2\ X\ A-2$, the final result, which the HP49G finds using again the rule of derivation of an expression multiplied by a constant. Again the opposite direction isn't possible. There is no command that collects expressions like:

$$\frac{\partial}{\partial X}\left(F(X)\right)+\frac{\partial}{\partial X}\left(G(X)\right)$$

to:

$$\frac{\partial}{\partial X}\left(F(X)+G(X)\right)$$

Here we could enter:

$$\frac{\partial}{\partial X}\left(F(X)\right)+\frac{\partial}{\partial X}\left(G(X)\right)$$

then list that contains the patterns to match:

$$\left\{\frac{\partial}{\partial \&v}\left(\&f\right)+\frac{\partial}{\partial \&v}\left(\&g\right)\quad\frac{\partial}{\partial \&v}\left(\&f+\&g\right)\right\}$$

and then use ↑ MATCH to convert the sum of the derivatives to the derivative of the sum. We keep these idea in mind for using them later for a program that does collection of differential forms. The rule for taking the derivative of a sum is known by the CAS of the HP49G if we build up the sum using $+$. But what about Σ ? Will it be able to handle for example

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N} X^n\right)$$

and convert it to

$$\sum_{n=1}^{N}\frac{\partial}{\partial X}\left(X^n\right)\ ?$$

Let's see. Enter:

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N} X^n\right)$$

and expand. The result is $d1\Sigma(n,1,N,X^2)$. Unfortunately it doesn't work this way. And this is a pity. The HP49G can do that. Enter $\sum_{n=1}^{N} X^n$, then X, and then press ∂. You get:

$$\frac{(X-1)\left(e^{(N+1)\,LN(X)}\left(N+1-\frac{1}{X}\right)\right)-\left(e^{(N+1)\,LN(X)}-X\right)}{SQ(X-1)}$$

Expand this to get:

$$\frac{\left((X-1)\,N+-1\right)|X|^{(N+1)}+X}{X^3-2\,X^2+X}$$

This result shows that the HP49G first found the result of $\sum_{n=1}^{N} X^n$ and then took the derivative of the result for X. First of all the question is why it doesn't do the same if we enter that in algebraic syntax? Then, what will happen if the sum can't be handled by the built-in CAS? To answer the second question, enter:

$$\sum_{n=1}^{N}\frac{X^n}{X-n}$$

then X, and then press ∂. After some seconds the HP49G returns:

$$d1\Sigma\left(n,1,N,\frac{X^n}{X-n}\right)$$

It can't find what the result of

$$\sum_{n=1}^{N}\frac{X^n}{X-n}$$

is and so it returns the formal derivative. There is no built in way to convert derivatives of sums in the form:

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N}\text{expression}(X,n)\right)$$

to:

$$\sum_{n=1}^{N}\frac{\partial}{\partial X}\left(\text{expression}(X,n)\right)$$

If the symbolic sum can be expanded to some other expression, then the derivative of this resulting expression will be taken. If we use the function ∂, this works only if we first enter the sum, then a name and then press ∂. Entering the whole thing at once, namely

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N}\text{expression}(X,n)\right)$$

will not work, even if the symbolic sum can be handled by the CAS.

On the other hand the command DERIV works in both RPL and algebraic syntax. The expression

$$\text{DERIV}\left(\sum_{n=1}^{N} X^n, X\right)$$

can be directly expanded to

$$\frac{((X-1)\ N-1)\ |X|^{(N+1)}+X}{X^3-2\ X^2+X}$$

The same result we get if we enter $\sum_{n=1}^{N} X^n$, then X, and then press DERIV and EXPAND. But again, expanding the expression

$$\text{DERIV}\left(\sum_{n=1}^{N} \frac{X^n}{X-n}, X\right)$$

we get

$$\text{d1}\Sigma\left(n,1,N,\frac{X^n}{X-n}\right)$$

because the CAS can't handle the sum $\sum_{n=1}^{N} \frac{X^n}{X-n}$. We can't somehow convert

$$\text{DERIV}\left(\sum_{n=1}^{N} \frac{X^n}{X-n}, X\right)$$

to

$$\sum_{n=1}^{N} \text{DERIV}\left(\frac{X^n}{X-n}, X\right)$$

using the built-in commands. So the rule for derivation of sums seems to be only implemented for + but not for Σ. The opposite direction, namely conversion of:

$$\sum_{n=1}^{N} \frac{\partial}{\partial X}(\text{expression}(X,n))$$

to:

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N} \text{expression}(X,n)\right)$$

is also not possible out of the box. In this case the HP49G takes the derivative of expression(X,n), and then tries to find the sum:

$$\sum_{n=1}^{N} \frac{\partial}{\partial X}(\text{expression}(X,n))$$

If it can handle this, it returns some result that doesn't contain any derivatives any more. If it can't find the sum, then it returns:

$$\sum_{n=1}^{N} \text{derivativeOfExpression}(X,n)$$

So we see that we have to program such conversions of sums built up with Σ. We have to watch out because if the variable of derivation is the same like the summation index, then we are not allowed to convert:

$$\frac{\partial}{\partial n}\left(\sum_{n=1}^{N}(\text{expression}(X,n))\right)$$

to:

$$\sum_{n=1}^{N}\left(\frac{\partial}{\partial n}(\text{expression}(X,n))\right)$$

(This is also true for the opposite direction.) So we have to check first if the derivation variable is different from the summation index, and do our conversions only if this condition is true.

Next rule of derivation is the product derivation rule:

$$\frac{\partial}{\partial X}(F(X)\ G(X)) = F(X)\ \frac{\partial}{\partial X}(G(X)) + G(X)\ \frac{\partial}{\partial X}(F(X))$$

Enter the derivative:

$$\frac{\partial}{\partial X}(F(X)\ G(X))$$

and evaluate it to get the result:

$$\frac{\partial}{\partial X}(F(X))\ G(X) + F(X)\ \frac{\partial}{\partial X}(G(X))$$

We see that the HP49G knows also this rule of differentiation. But here we start suspecting what one of the main problems of our program will be, that collects differential forms. We have to do it in such a way, that it will be able to collect:

$$\frac{\partial}{\partial X}(F(X))\ G(X) + F(X)\ \frac{\partial}{\partial X}(G(X))$$

but also:

$$G(X)\ \frac{\partial}{\partial X}(F(X)) + F(X)\ \frac{\partial}{\partial X}(G(X))$$

or any equivalent form, to:

$$\frac{\partial}{\partial X}(F(X)\ G(X))$$

That means that we have to somehow make sure that it will do its work with arbitrary differential forms, and not only with differential forms written in some particular way. Let's have a particular example. Enter:

$$\frac{\partial}{\partial X}(X^2\ \text{SIN}(X))$$

and evaluate once to get the result:

$$\frac{\partial}{\partial X}(X^2)\ \text{SIN}(X) + X^2\ \frac{\partial}{\partial X}(\text{SIN}(X))$$

The rule of derivation of products was used once. Press EVAL again to get the result $2\ X\ \text{SIN}(X) + X^2\ \text{COS}(X)$.

Now we move on to the derivation rule of powers:

$$\frac{\partial}{\partial X}(F(X)^n) = n\ F(X)^{n-1}\ \frac{\partial}{\partial X}(F(X))$$

If you enter:

$$\frac{\partial}{\partial X}(F(X)^n)$$

and evaluate, you get the result:

$$n\ F(X)^{n-1}\ \frac{\partial}{\partial X}(F(X))$$

The opposite direction is again not possible, there is no available command to collect the expression:

$$n\ F(X)^{n-1}\ \frac{\partial}{\partial X}(F(X))$$

to:

$$\frac{\partial}{\partial X}\left(F(X)^{n}\right)$$

For an example enter:

$$\frac{\partial}{\partial A}\left(A^{4}\right)$$

and press EVAL to get $4\ A^{3}$.

We take a look to the derivation rule for quotients:

$$\frac{\partial}{\partial X}\left(\frac{F(X)}{G(X)}\right)=\frac{G(X)\ \frac{\partial}{\partial X}(F(X))-F(X)\ \frac{\partial}{\partial X}(G(X))}{G(X)^{2}}$$

Enter:

$$\frac{\partial}{\partial X}\left(\frac{F(X)}{G(X)}\right)$$

and evaluate. You get the result:

$$\frac{G(X)\ \frac{\partial}{\partial X}(F(X))-F(X)\ \frac{\partial}{\partial X}(G(X))}{SQ(G(X))}$$

which shows that also this rule is known to the CAS. But again re-collecting the result to:

$$\frac{\partial}{\partial X}\left(\frac{F(X)}{G(X)}\right)$$

is not possible. To have an example, enter:

$$\frac{\partial}{\partial X}\left(\frac{SIN(X)}{X}\right)$$

and evaluate to get:

$$\frac{X\ \frac{\partial}{\partial X}(SIN(X))-SIN(X)}{SQ(X)}$$

Next evaluation gives:

$$-\frac{SIN(X)-X\ COS(X)}{X^{2}}.$$

Next comes the "chain rule", the rule for taking derivatives of nested functions:

$$\frac{\partial}{\partial X}(F(G(X))) = \frac{\partial}{\partial G(X)}(F(G(X)))\ \frac{\partial}{\partial X}(G(X))$$

Enter:

$$\frac{\partial}{\partial X}(F(G(X)))$$

and evaluate to get $d1G(X)\ \ d1F(G(X))$. Here the HP49G returned the result using $d1$-notation, but the result is nonetheless correct. Remember, $d1F(G(X))$ is the derivative of $F(G(X))$ for $G(X)$. Let's have an example again. We use $G(X) = X^2$ and $F(G(X)) = SIN(G(X)) = SIN(X^2)$. Enter:

$$\frac{\partial}{\partial X}(SIN(X^2))$$

and press EVAL to get:

$$COS(X^2)\ \frac{\partial}{\partial X}(X^2)$$

The HP49G has found the "outer" derivative:

$$\frac{\partial}{\partial X^2}(SIN(X^2)) = COS(X^2)$$

and multiplied this with the "inner" derivative:

$\frac{\partial}{\partial X}(X^2)$, giving the result:

$$COS(X^2)\ \frac{\partial}{\partial X}(X^2)$$

If you press EVAL once more, you get $2\ X\ COS(X^2)$ (i.e. rule of derivation of powers). As you might have expected, the is no available command for collecting:

$$\frac{\partial}{\partial G(X)}(F(G(X)))\ \frac{\partial}{\partial X}(G(X))$$

to:

$$\frac{\partial}{\partial X}(F(G(X))).$$

As we see all the above rules of differentiation are known to the CAS of the HP49G, but only in one direction, namely in the "expanding" direction. The opposite, call it "collecting" direction, is not possible. However, sometimes it is quite useful to have that feature of collection of differential forms. (We will see that this can be a quite a help for solving some types of differential equations.) So we are going to make a program for this. Let it be said here, that the program will not do miracles. In many cases it will fail to collect the differential forms, though they could be collected. However, it will do its work in many other cases. In addition it will demonstrate some techniques of programming and dealing with algebraic objects, which I hope will generate appetite for more ideas and further improvements. Before we continue, deactivate the step by step feature, as we don't want stepwise evaluation of derivatives any more.

Let's consider first some general things about the program. In order to collect differential forms, it will make heavy use of the pattern matching commands ↑ MATCH and ↓ MATCH. But here we have the first problem. The commands can't be used with differential forms written in dn notation. Imagine for example that we have $d1F(X) + d1G(X)$,

which can be collected to:

$$\frac{\partial}{\partial X}(F(X)+G(X))$$

First of all, the undefined functions $F(X)$ and $G(X)$ could also be $H(Y)$ and $P(Z)$. That means that we have to use the pattern matching commands with general patterns. But since the differential forms are written as $d1F(X)$ and $d1G(X)$, the names of the functions alone are not available for pattern matching. We only have the names $d1F$ and $d1G$. Matching $d1F(X)$ to $\frac{\partial}{\partial X}(F(X))$ is only possible for this particular case, namely using the pattern matching list:

$$\begin{bmatrix} d1F(X) & \frac{\partial}{\partial X}(F(X)) \end{bmatrix}$$

We can't use any other more general pattern that would also match for example $d1G(Y)$ to $\frac{\partial}{\partial Y}(G(Y))$. We can't use the pattern matching list:

$$\begin{bmatrix} d1F(X) & \frac{\partial}{\partial X}(F(X)) \end{bmatrix}$$

This would of course match $d1G(Y)$ to $\frac{\partial}{\partial Y}(G(Y))$ and $d1F(X)$ to $\frac{\partial}{\partial X}(F(X))$, but it would also match $F(X)$ to $\frac{\partial}{\partial X}(F(X))$ because, as already said, for the calculator both $d1F(X)$ and $F(X)$ are the same general pattern. (Remember, $d1F$ is just a name.) In addition, the expression $d1F(X)$ doesn't contain the variable of differentiation in a way that makes it easy to do pattern replacement. So the first requirement is that we have to convert first all derivatives written in $d1$ notation to derivatives written in $\frac{\partial}{\partial}$ notation. And for this we have to do parsing. Since it could be useful to have such a program as a stand alone utility for converting expressions with $d1$ derivatives to expressions with $\frac{\partial}{\partial}$ derivatives, we are going to program that separately. Then, the program that will do collection of differential forms, can just call this stand alone utility and use it for the conversion of $d1$ derivatives to $\frac{\partial}{\partial}$ derivatives. Let's take a look at the listing of the program $dn \rightarrow dv$ which takes an expression from stack level 1, and returns it with all $d1$ derivatives converted to $\frac{\partial}{\partial}$ derivatives.

```
<<
  PUSH                      @Save user flags
  -100. CF                  @No step by step
  "Converting ∂ to d"       @Display message
  1 DISP
  IF
    EXPAND LVAR {} ≠        @If expression has
                            @rational variables
    OVER TYPE 9. == AND     @and it is an algebraic
                            @object
  THEN                      @then:
    "Filtering names out"   @Display message
    1 DISP
    LVAR AXL                @Return list of rat. vars
    {}
    → varl                  @Store empty list in local
    <<                      @Local var. procedure
      1.
      <<                    @Start of DOSUBS procedure
        IF                  @for all rational variables
          DUP TYPE 6.       @If rat var. is a name
          ==
        THEN                @then drop it
          DROP
```

```
    ELSE              @else
      'varl' STO+     @ add it to list varl.
    END
  >>
  DOSUBS              @Do to all rat. vars.
  varl                @Return varl
>>
IF
  DUP {} ≠            @If result is not empty
THEN                  @then we parse all rat.
  1.                  @variables
  <<                  @Start of DOSUBS procedure
    DUP →STR          @Make a copy, conv. to str.
    2. OVER SIZE      @Substr. chars. 2 to length
    1 - SUB           @of string - 1
    {} ""             @Store in locals
    → dFormAlg dFormStr
    dvars dTemp
    <<
      1. SF           @Set flag 1 (indicator)
      DO              @Do
        "Searching diff. forms"
        1. DISP       @Display message
        IF            @If
          dFormStr    @string starts with "d("
          "d(" POS
          1. ==
        THEN   @then
          "No diff. forms"
          1. DISP     @Display message
          1. CF       @Clear flag 1
        ELSE          @else
          IF          @if
            dFormStr
            HEAD      @If first char. is "d"
            "d" ==
          THEN        @then
            "Diff. form found"
            1. DISP   @Display message
            "d"       @Store "d" in dTEMP
            'dTemp'
            STO
            dFormStr  @Store tail of dFormStr
            TAIL      @in dFormStr
            'dFormStr'
            STO
          ELSE        @else
            "No diff. forms"
            1. DISP   @Display message
            1. CF     @Clear flag 1
          END
      IF              @If flag 1 is set
        1. FS?
      THEN            @then
        "Parsing diff. form"
        1. DISP       @Display message
        WHILE         @While
          dFormStr    @dFormStr starts with number
          HEAD DUP    @≥ 0 and ≤ 9
          "0" ≥
          SWAP "9"
          ≤ AND
          dFormSTr    @and it has an opening
          "(" POS     @parenthesis at a position
          3. ≥        @≥ 3
          AND         @(We check pattern "dn(")
        REPEAT        @repeat
          'dtemp'     @add first char of dFormStr
          dFormStr    @to dTemp
          HEAD STO+
          dFromStr    @put the rest in dFormStr
          TAIL
          'dFormStr'
          STO
        END
        IF            @If dFormStr has "(" in
          dFormStr    @first or second position
          "(" POS
          2. ≤
        THEN          @then
          IF          @If
            dFormStr  @dFormStr doesn't start
            "0" ≥     @with char. between 0
            dFormStr  @and 9
```

```
        "9" ≤
        AND NOT
      THEN      @then
        'dvars'   @add object that results
        dTemp     @from substr 1 to length
        2 OVER    @-1 from dTemp
        SIZE      @to dvars
        SUB OBJ→
        STO+
      ELSE      @else
        1. CF   @clear flag 1
      END
    ELSE        @else
      'dvars'   @add object that results
      dTemp 2   @from substr 1 to length -1
      OVER      @from dTemp to dvars
      SIZE SUB
      OBJ→
      STO+
    END
  END
END
UNTIL
  1 FC?         @until flag 1 is clear
END
IF
  dvars {} ≠    @dvars contains something
THEN
  dFormAlg
  dvars REVLIST @Reverse dvars list
  IF            @If
    dFormStr    @dFormStr starts with char.
    HEAD DUP    @between "0" and "9"
    "0" ≥
    SWAP "9"
    ≤ AND
  THEN          @then
    "d"         @add "d" at start of
    dFormStr +  @dFormStr
  ELSE          @else
    dFormStr    @return dFormStr
  END
```

```
      "Build-up MATCH list"
      1. DISP       @Display message
      DUPDUP "("    @Find number of deriv. var.
      POS 1. +      @Make list {nums deriv vars}
      OVER ")"
      POS 1 -
      SUB "{" SWAP
      + "}" + OBJ→
      → vars          @Store local
      <<
        SWAP 1.
        <<          @DOSUBS procedure
          "∂" vars @Make str "∂var(expr)"
          ROT GET
          + "(" +
          SWAP +
          ")" +
        >>
        DOSUBS      @Do with all list elements
        "'" SWAP +  @Add quotes and make alg.
        "'" + OBJ→
        2 →LIST     @Make pattern match list
      >>
    END
  >>
>>
DOSUBS              ©Do with all list elements
IF                  @If we don't have an alg.
  DUPDUP TYPE 9. ≠  @or name
  SWAP TYPE 6. ≠
  AND
THEN                @then
  "MATCHing diff. forms"
  1. DISP           @display message
  1.
  <<
    ↑MATCH DROP
  >>
  DOSUBS            @MATCH every pattern
END
ELSE                @else
```

```
      DROP                    @Drop
    END
  END
  POP                         @Restore user settings
>>
```

The program dn → dv which comes with this document (he, he, no need to type it ;-)) is really no easy thing to read and to understand how and why it works. I have made corrections and corrections of corrections until it worked. So much of it is kind of patchwork. And it isn't perfect also. For example it will crash if you give it d100F(X) because it will try to find out what the hundredth rational variable (inside the parentheses) of F(X) is. But it will work correctly if you give it expressions that are convertible to the notation $\frac{\partial}{\partial}$. If no derivative is contained in the expression that you give it, it returns the expanded expression. Should we try it? But of course! Enter d1d1F(X) − d1G(Y) d2F1(X,Y) and press dn→dv. The HP49G displays some messages (that Nick finds informative ;-)) and then it returns:

$$-\left(\frac{\partial}{\partial Y}\left(G(Y)\right)\ \frac{\partial}{\partial Y}\left(F1(Y,Y)\right)-\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right)\right)$$

It works! (Even I wouldn't expect that ;-))

Let's comment some of the used techniques before we proceed, as they seem to be interesting. First of all, we have that EXPAND at the beginning of the program. Is it necessary? Well, the expression that contains differential forms, could contain them in any possible notation. For example, we could give the program:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right)$$

or:

$$\frac{\partial}{\partial X}\left(d1F(X)\right)$$

or:

d1d1F(X). We expand at the beginning in order to convert the expression to the standard fully expanded form. This way we are able to know later on in the program, that the expression is in a particular form, no matter how it was entered.

Another thing that should be noticed is how the list of rational variables is constructed. Take for example the derivative d1F(X,Y,Z). The program converts that to a string and extracts the sub string "X,Y,Z". Then the list delimiters are added and the string is converted to "{X,Y,Z}". This string is then converted to a list using the command OBJ→. Notice here the commas between the list elements. The sequence "{X,Y,Z}" OBJ→ is exactly the same as if you have entered the list {X,Y,Z} from the command line. Normally list elements are separated by spaces but commas will also work. You can even enter X,Y,Z, which is exactly the same like entering X Y Z and creates the three names in stack levels 3 to 1. In this case it is very helpful not having to replace the commas in the string to spaces before using OBJ→. (However, if we had to do that, we could use the command SREPL.)

The last thing that seems remarkable is that the same comparison commands, like for example >, ≥ and so on, can be used to compare… characters! Of course we don't compare the characters themselves but rather their character number in the character table of the HP49G. This is very helpful for alphabetising purposes, or for finding if some string starts with a character, the character number of which is within a certain range.

Now that we have dn → dv, let's proceed and see what else we need. The collection of differential forms can be very hard to program, if we don't define the standard form in which the algebraic form has to be delivered to our collecting algorithms. Of course we have already used EXPAND to bring the expression in its standard fully expanded form, but this is not enough. The command EXPAND will sometimes return sums of products, sometimes sums of terms, and so on. We can't predict what the general form of its results will be. So we make a convention. The expression has to be delivered to our collecting algorithm as a sum of products. This way we will know that it has the form $T_1 + T_2 + \ + T_n$, where $T_1, T_2, \ , T_n$ are all products (i.e. they don't contain any sums). The HP49G has the command FDISTRIB, which fully distributes and / over + and −. This command has also the *big* advantage, that it returns sums but no differences. For example, entering

$$\frac{A-B}{C}$$

and using FDISTRIB, returns:

$$\frac{A}{C}+-\frac{B}{C}$$

(i.e. the sum of $\frac{A}{C}$ and $-\frac{B}{C}$ rather than the difference of $\frac{A}{C}$ and $\frac{B}{C}$). Don't underestimate this simple fact. If we had sometimes a sum and sometimes a difference, then using the summands one after the other in a program would be much more complicated because we would have to check if they are connected by + or by −. If we have the expression in its fully distributed form, then we can examine each summand for itself, and decide if in some of them the differential forms can be collected. Then we can add them pair wise and examine every possible pair $T_i + T_j$, where $i \neq j$, with pattern matching of the differential forms. We need a program that returns all summands of some expression separately and unfortunately the HP49G doesn't have such a command. But fortunately the Sequences, Series and Limits Marathon was done before the Basic Calculus Marathon. (There is method in my insanity ;-)) In that marathon we had the program → TERMS that does exactly this. It returns a list with all summands of some expression. If you don't have that marathon then it doesn't matter, because the program → TERMS comes also with this marathon.

Let's see now the listing of the program dCOLLECT.1, which (hopefully) collects differential forms. It uses all the ideas from above and also the pattern matching commands quite often. It needs d1 → dv and → TERMS to run, so you should have these programs at the same directory. The program is way from being perfect, but as already said, all programs of the marathon are for demonstrating programming techniques and creating appetite for improvement.

```
<<
  PUSH                      @Save user's settings
  dn → dv                   @Convert to ∂ notation
   → TERMS                  @return list of terms
   →  diffTerms             @Store in local
  <<
    1. diffTerms SIZE       @Do for each term
    FOR I
      "Checking " I + " term
" + 1. DISP                 @Display message
      diffTerms I GET       @Extract term from list
      FACTORS               @Return list of factors and
      { 1. 1.} {}           @multiplicities
       →  termFacts
      facts difFacts        @Store in locals
      <<
        termFacts 1.
        <<                  @DOSUBS procedure starts
          IF                @If we have an element at
            NSUB 2. MOD     @an even position
            NOT
          THEN              @then it is a power
            2.  → LIST      @so make list {factor power}
          END
        >>
```

```
        DOSUBS              @Now the list is in the form
                            @{{fact1 pow1}{fact2 pow2}…}
        "Separating ∂"      @Now we are going to
        1. DISP             @separate diff. forms from
                            @other factors
        1.
        <<                  @Another DOSUBS procedure
          IF                @If
            DUP HEAD
            {'∂&V(&A)' '∂&V(&A)'}
            ↓MATCH NIP      @factor is a diff. form
          THEN              @then convert it to a list
            1. →LIST
            'difFacts'      @and enter 'difFacts
          ELSE              @else
            'facts'         @enter 'facts'
          END
          SWAP STO+         @Add factor to approp. list
        >>
        DOSUBS      @Now we have all factors that
                    @are diff. forms in difFacts.
                          @All the others are in facts
        IF                  @If there were more than one
          difFacts SIZE     @diff. forms
          1. >
        THEN                @Then we take the highest
          'facts'   @derivative and add the others
                          @to the common factors
          "Isolate highest diff.
" 2. DISP                  @Message
          difFacts TAIL     @Add all but the highest
                            @der.
          1.                @to common factors
          << OBJ→ DROP >>
          DOSUBS STO+
          difFacts HEAD     @Put highest derivative
          1. →LIST          @in difFacts
          'difFacts' STO
        END
        IF                  @If difFacts isn't empty
          difFacts {} ≠
THEN                  @then
  "(∂)^n → ∂*(∂)^(n-1)"
  2. DISP             @We will convert all derivs.
                      @from (∂X(F(X)))^n to
                      @∂X(F(X))*(∂X(F(X)))^(n-1)
                      @(See explanations after
                      @program listing.)
  difFacts 1.
  <<                  @Yet another DOSUBS proc.
    OBJ→ DROP         @Convert { deriv pow } to
    OVER SWAP         @{ deriv deriv pow-1 }
    1. - R→I
    3. →LIST
  >>
  DOSUBS
  'difFacts' STO      @Store in difFacts
  "Collecting ∂ of powers"
  2. DISP             @Another message
  difFacts 1.
  <<                  @DOSUBS proc.
    IF                @If
      facts OVER      @Sub expression F(X)
      HEAD OBJ→       @of ∂X(F(X))
      3. DROPN        @appears in the factors
      POS DUPDUP      @that are not derivatives
    THEN              @then
      facts SWAP      @get factor and its power n
      GET facts
      ROT 1 +
      GET
       → factor       @Store locally
      power
      <<
        "'∂"          @Construct ∂X(F(X)^n+1)
        OVER HEAD
        OBJ→ DROP2
        NIP + "(" +
        factor power
        1. + R→I
        IF
          DUP 0. <
```

```
        THEN
          NEG ^ INV
        ELSE
          ^
        END
        →STR 2. OVER
        SIZE 1. - SUB +
        ")'" + OBJ→
        1. SWAP   @Put ∂X(F(X)^n+1) in first
        PUT       @place of difFact
        'facts'   @Construct {F(X) -n n+1 -1}
        factor
        power
        NEG power
        1. + R→I
        -1. 4. →LIST
        STO+      @Add to list facts
      >>
    ELSE          @else
      DROP2       @drop unnecessary objects
    END
  >>
  DOSUBS          @Do to every diff. Factor
  'difFacts' STO  @store in difFacts
END
'difTerms'
I facts 1.
<<                @DOSUBS proc for each factor
  IF              @If we are at an even pos.
    NSUB 2. MOD
    NOT
  THEN            @then
    R→I ^         @we raise to the power
  END
>>
DOSUBS
1 + ∏LIST         @Add 1 to the list and
                  @make prod. of list elements
EXPAND dn→dv      @Use dn→dv again
IF
  difFacts {} ≠   @If there are diff. facts
  THEN            @then
    difFacts 1.
    <<            @Product of diff. factors
      OBJ→ DROP
      R→I ^ *
    >>
    DOSUBS
    1 +
    ∏LIST         @Make product of derivs.
    *             @Mult. factors and derivs.
  END
  PUT             @Put back in orig. list
>>
NEXT
1. SF             @Flag 1 is our indicator
WHILE             @While more than one term
  diffTerms SIZE  @and flag 1 is set
  1. > 1 FS? AND
REPEAT            @repeat
  1. diffTerms SIZE
  1. -
  FOR I           @for I=1 to number of diff.
                  @terms - 1
    I 1 +
    diffTerms SIZE  @for J=2 to number of diff.
    FOR J         @terms
      "Diff. terms "
      I + " " +   @Construct and display
      J + 1. DISP @message
      diffTerms   @Make sum of Ith and Jth
      I GET       @term
      diffTerms
      J GET +
      → dTerm     @Store in local
      <<
        "A*∂B+B*∂A → ∂(A*B)"
        2. DISP   @Message
        dTerm
        CASE      @Use product rule
             { '&A*∂&V(&B)+&B*∂&V(&A)'
             '∂&V(&A*&B)' }
```

```
      ↑MATCH   @Prod. patt. match
    THEN
      1.     @return 1.
    END      @Same      match      written
different.
      { '&A*∂&V(&B)+∂&V(&A)*&B'
      '∂&V(&A*&B)' }
      ↑MATCH
    THEN
      1.
    END
      COLLECT   @Collect     and     retry
matches
      { '&A*∂&V(&B)+&B*∂&V(&A)'
      '∂&V(&A*&B)' }
      ↑MATCH
    THEN
      1.
    END
      { '&A*∂&V(&B)+∂&V(&A)*&B'
      '∂&V(&A*&B)' }
      ↑MATCH
    THEN
      1.
    END      @If no match, return 0.
    0.
END
"C*∂(A) → ∂(C*A)"
2. DISP     @Message
SWAP
{ '&C*∂&v(&v)' '∂(&v*&f)'
  'NOT POSNAME(&C,&v)' }
↑MATCH      @Match mult. with constant.
ROT OR
"∂A+∂B → ∂(A+B)"
2. DISP     @Message
SWAP        @Match sums
{'∂&V(&A)+∂&V(&B)'
  '∂&V(&A+&B)' }
↑MATCH ROT OR
SWAP
{'∂&V(&A)-∂&V(&B)'
  '∂&V(&A-&B)' }
↑MATCH ROT OR
IF          @If we had a match
THEN
  1. SF     @Set flag 1
  diffTerms    @Replace the two terms
  1. I 1. -    @of diffTerms that we used
  SUB          @with the matched one
  diffTerms
  I 1. +
  OVER SIZE SUB
  DUP 1. J.
  diffTerms
  SIZE 4. PIC
  SIZE - - 1. -
  SUB SWAP J
  diffTerms
  SIZE PICK3 SIZE
  - - 1. +
  OVER SIZE SUB
  + + +
  'diffTerms' @Store in diffTerms
  STO
  diffTerms    @Store numbers > than
  SIZE I +     @end of FOR loop
  DUP 'I' STO  @in I and J to exit
  'J' STO      @the FOR loops
ELSE           @Else (no match)
  DROP         @Drop terms
  1. CF        @Clear flag 1
END
    >>
  NEXT
NEXT
END                    @End of WHILE
diffTerms 0 +
ΣLIST                  @Sum of all terms
{ '&C*∂&v(&v)' '∂(&v*&f)'
```

```
      'NOT POSNAME(&C,&v)' }
      ↑MATCH              @Match mult. with constant.
      DROP
    >>
    POP
>>
```

There are many interesting things in this program, but I guess you are rather interested to see it in work first. So, let's have some tests and then we can discuss what the program does and why.

In all the following examples I use $\frac{\partial}{\partial}$ notation but the d1 notation could be used as well. The program takes differential forms in any of the two notations or even mixed up. We start with an easy example. Enter:

$$F(X)\ \frac{\partial}{\partial X}(F(X))$$

According to the rule for differentiating powers and to the chain rule we have:

$$\frac{\partial}{\partial X}\left(F(X)^2\right) = 2\ F(X)\ \frac{\partial}{\partial X}(F(X)) \Leftrightarrow F(X)\ \frac{\partial}{\partial X}(F(X)) = \frac{\frac{\partial}{\partial X}\left(F(X)^2\right)}{2}$$

Since

$$F(X)\ \text{d1}F(X) = F(X)\ \frac{\partial}{\partial X}(F(X))$$

the program dCOLLECT.1 should return

$$\frac{\frac{\partial}{\partial X}\left(F(X)^2\right)}{2}$$

Press dCOLLECT.1. In about 15 seconds the HP49G returns:

$$\frac{\partial}{\partial X}\left(\frac{1}{2}\ F(X)^2\right)$$

Indeed it worked!

Enter:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(G(X))\right) + G(X)\ \frac{\partial}{\partial X}(F(X)) + F(X)\ \frac{\partial}{\partial X}(G(X))$$

Press ENTER to make a copy and then press dCOLLECT.1. After about 1 minute and 10 seconds in agony the HP49G returns the result:

$$\frac{\partial}{\partial X}\left(G(X)\ F(X) + \frac{\partial}{\partial X}(G(X))\right)$$

This is the same expression with all differential forms collected in one derivative. But is that really the same like what we entered? Press - and expand to get a fat 0, which shows that the two expressions were indeed equal to each other.

Enter:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right) + \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(G(X))\right) + \frac{\partial}{\partial X}(G(X))$$

and press dCOLLECT.1. After about 47 seconds of dancing messages at the top of the screen the HP49G returns:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)+G(X)\right)+G(X)\right)$$

Enter:

$$\left(3\ F(X)^2+G(X)^2\right)\ \frac{\partial}{\partial X}\left(F(X)\right)+2\ G(X)\ F(X)\ \frac{\partial}{\partial X}\left(G(X)\right)$$

and press dCOLLECT.1 again. This time the HP49G needs about 1 minute and 8 seconds, and returns the collected form:

$$\frac{\partial}{\partial X}\left(F(X)\ G(X)^2+F(X)^3\right)$$

Enter:

$$F(X)^2\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right)+2\ F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)^2$$

Press dCOLLECT.1 to get the partially collected form:

$$\frac{\partial}{\partial X}\left(F(X)^2\ \frac{\partial}{\partial X}\left(F(X)\right)\right)$$

in about 39 seconds. The result could be further collected to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{3}\ F(X)^3\right)\right)$$

After the examples we will see why the program didn't collected completely.

Enter:

$$2\ F(X)^2\ \frac{\partial}{\partial X}\left(F(X)\right)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right)+2\ F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)^3$$

and press again dCOLLECT.1. The HP49G returns:

$$\frac{\partial}{\partial X}\left(F(X)^2\ \frac{\partial}{\partial X}\left(F(X)\right)^2\right)$$

in about 46 seconds. Equivalent forms of this result, having differential forms collected differently, are:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{3}\ F(X)^3\right)\right)\ \frac{\partial}{\partial X}\left(F(X)\right)$$

and:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)^3\right)\ \frac{\partial}{\partial X}\left(F\left(\frac{1}{3}\right)\ X\right)\right)$$

We will see how to get the second or the third form after the examples.

Enter:

$$F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right)+\frac{\partial}{\partial X}\left(F(X)\right)^2+\frac{\partial}{\partial X}\left(F(X)\right)$$

and press dCOLLECT.1. You get:

$$\frac{\partial}{\partial X}\left(F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)+F(X)\right)$$

in about 46 seconds. Here again the result could be further collected to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{2}\right)\ F(X)^2\right)+F(X)$$

Enter:

$$(F(X)+1)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)+\frac{\partial}{\partial X}(F(X))^2$$

and press dCOLLECT.1 to get:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\ F(X)\right)+\frac{\partial}{\partial X}(F(X))$$

in about 46 seconds. In this case the result could also have been collected further to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{2}\right)\ F(X)^2\right)+\frac{\partial}{\partial X}(F(X))$$

or to:

$$\frac{\partial}{\partial X}\left((F(X)+1)\ \frac{\partial}{\partial X}(F(X))\right)$$

After the examples we will see how to get these forms using dCOLLECT.1 or COLLECT.

Enter:

$$F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)+\frac{\partial}{\partial X}(F(X))^2$$

Pressing dCOLLECT.1 returns the collected form:

$$\frac{\partial}{\partial X}\left(F(X)\ \frac{\partial}{\partial X}(F(X))\right)$$

after 27 seconds. Also here we could go further and get:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{2}\right)\ F(X)^2\right).$$

Enter:

$$\frac{2\ G(X)\ F(X)\ \frac{\partial}{\partial X}(F(X))-F(X)^2\ \frac{\partial}{\partial X}(G(X))}{G(X)^2}$$

The program dCOLLECT.1 needs about 49 seconds to return:

$$\frac{\partial}{\partial X}\left(\frac{1}{G(X)}\ F(X)^2\right).$$

Enter:

$$\frac{G(X)\ \frac{\partial}{\partial X}(F(X))-\left(F(X)-G(X)^2\right)\ \frac{\partial}{\partial X}(G(X))}{G(X)^2}$$

and press dCOLLECT.1 to get:

$$\frac{\partial}{\partial X}\left(\frac{1}{G(X)}\ F(X)+G(X)\right)$$

after about 57 seconds.

Last example. Enter:

$$3\ F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)^2 - 3\ F(X)^2\ \frac{\partial}{\partial X}\left(F(X)\right)^2$$

Press dCOLLECT.1 to get:

$$\frac{\partial}{\partial X}\left(3\ F(X)^2 - 2\ F(X)^3\right)\ \frac{\partial}{\partial X}\left(F(X)\right)$$
$$2$$

in about 1 minute.

As we see the program works in many cases. In many other cases it returns only partially collected results. And I am sure that you will find even more cases in which it doesn't work at all or it even crashes. Feel free to change its code and make it better, if you wish. But to make it better, some details about its inner workings are necessary.

We start explaining the cases where the program gives only partially collected results, and we try to find a way to make the collection of differential forms complete in these cases. For example why was

$$F(X)^2\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right) + 2\ F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)^2$$

transformed to

$$\frac{\partial}{\partial X}\left(F(X)^2\ \frac{\partial}{\partial X}\left(F(X)\right)\right)$$

but not to

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{3}\ F(X)^3\right)\right) ?$$

Let's follow what the program did. The second term of the original expression was transformed to:

$$\frac{\partial}{\partial X}\left(F(X)^2\right)\ \frac{\partial}{\partial X}\left(F(X)\right)$$

using the rule of differentiation of powers. This happened at the point where the message Collecting ∂ of powers was displayed. When the program displayed A ∂B + B ∂A → ∂(A B), the rule of differentiation was used, and the sum

$$F(X)^2\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right) + \frac{\partial}{\partial X}\left(F(X)^2\right)\ \frac{\partial}{\partial X}\left(F(X)\right)$$

was transformed to

$$\frac{\partial}{\partial X}\left(F(X)^2\ \frac{\partial}{\partial X}\left(F(X)\right)\right)$$

Now, from this point on, the rule of differentiation of powers could be used again on the sub expression:

$$F(X)^2\ \frac{\partial}{\partial X}\left(F(X)\right)$$

And this is exactly what the program doesn't do. It doesn't check if using some differentiation rule results in a derivative, in which the expression that is differentiated can itself be further collected. Can we do something to achieve complete collection of differential forms in this case? Let's consider first an interactive possibility. What would happen if we had the expression

$$F(X)^2\ \frac{\partial}{\partial X}\left(F(X)\right)$$

on the stack and we used dCOLLECT.1? Enter:

$$F(X)^2 \quad \frac{\partial}{\partial X}(F(X))$$

and press dCOLLECT.1 to get:

$$\frac{\partial}{\partial X}\left(\frac{1}{3}F(X)^3\right)$$

The expression was collected. Of course it would be a tedious task to have the result

$$\frac{\partial}{\partial X}\left(F(X)^2 \quad \frac{\partial}{\partial X}(F(X))\right)$$

on stack, use OBJ → to explode it, then use dCOLLECT.1 on the sub expression:

$$F(X)^2 \quad \frac{\partial}{\partial X}(F(X))$$

to transform it to:

$$\frac{\partial}{\partial X}\left(\frac{1}{3}F(X)^3\right)$$

and then recombine this result and the rest of the objects returned by OBJ → to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{3}F(X)^3\right)\right)$$

But we don't need to do that. Since we are talking about sub expressions of a given expression, the EQW comes into mind. There we can select some sub expression, apply some built-in command on that sub expression only, and get the result that we want to have. If there would be some way to apply our own programs on some particular sub expression, then we could take

$$\frac{\partial}{\partial X}\left(F(X)^2 \quad \frac{\partial}{\partial X}(F(X))\right)$$

in the EQW, select the sub expression

$$F(X)^2 \quad \frac{\partial}{\partial X}(F(X))$$

and use dCOLLECT.1 to transform it to

$$\frac{\partial}{\partial X}\left(\frac{1}{3}F(X)^3\right)$$

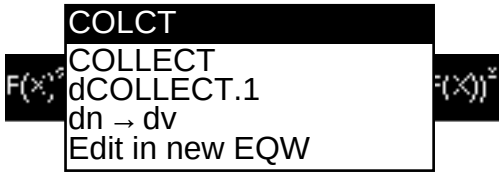
And guess what? There is a way. (Or else why should Nick tell all this? ;-)). We can make a user menu that is active when we are in the EQW. In that menu we can put all things that we need. Let's see how we do that. When the EQW is active and you have selected some sub expression, pressing CUSTOM has a special meaning. Under these conditions the HP49G checks if a program named STARTEQW exists in the current path. If it does, then this program is executed. Now, we can use this capability to display a pop up menu that contains all things we need, including dCOLLECT.1. Actually we can add anything we want in that menu, provided that the objects contained in that menu, need one algebraic object as input and return one algebraic object as output. (Or anything that is allowed in algebraic objects.) This is one of the many great ideas of VPN for which I am very grateful. Let's make the program STARTEQW. Consider the program on the next page.

```
<<
  ""                          @Pop-up has no title
  {
    COLCT                     @First menu item is the old
                              @command COLCT, which
                              @sometimes collects
                              @differently than COLLECT.
    COLLECT                   @Second comes COLLECT.
    dCOLLECT.1                @Then comes dCOLLECT.1.
    dn → dv                   @Then comes dn → dv
    { "Edit in new EQW"       @Then comes a menu item that
      << EQW >>               @shows "Edit in new EQW" but
    }                         @executes << EQW >> when
                              @selected.
  }
  1.                          @We display the pop-up with
  CHOOSE                      @the first item selected
  IF                          @If user pressed [ENTER]
  THEN                        @then we evaluate (execute)
    EVAL                      @the selected item
  END
>>
```

Store that in STARTEQW. (Or simply use the program that comes with this document.) Let's see that in action. Go to the EQW and enter:

$$F(X)^2 \quad \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))^2\right)+2 \quad F(X) \quad \frac{\partial}{\partial X}(F(X))^2$$

again. Select the whole expression. Now the expression is displayed inverse. Press CUSTOM to start the program STARTEQW. The pop-up menu is displayed over the selected expression and the screen of the calculator looks like the picture to the right. Press ▼ twice to select the item dCOLLECT.1, and then press ENTER. Wow! The program dCOLLECT.1 runs just as if it was one of the built-in commands. It displays its messages and works just like it does when we use it from the stack. In fact, behind the scenes it does work on the stack. First it puts the selected sub expression on the stack. Then the selected menu item takes the sub expression, does its work with it, and returns its output on the stack. Then the result is taken back to the EQW, where it replaces the originally selected sub expression. So now the result

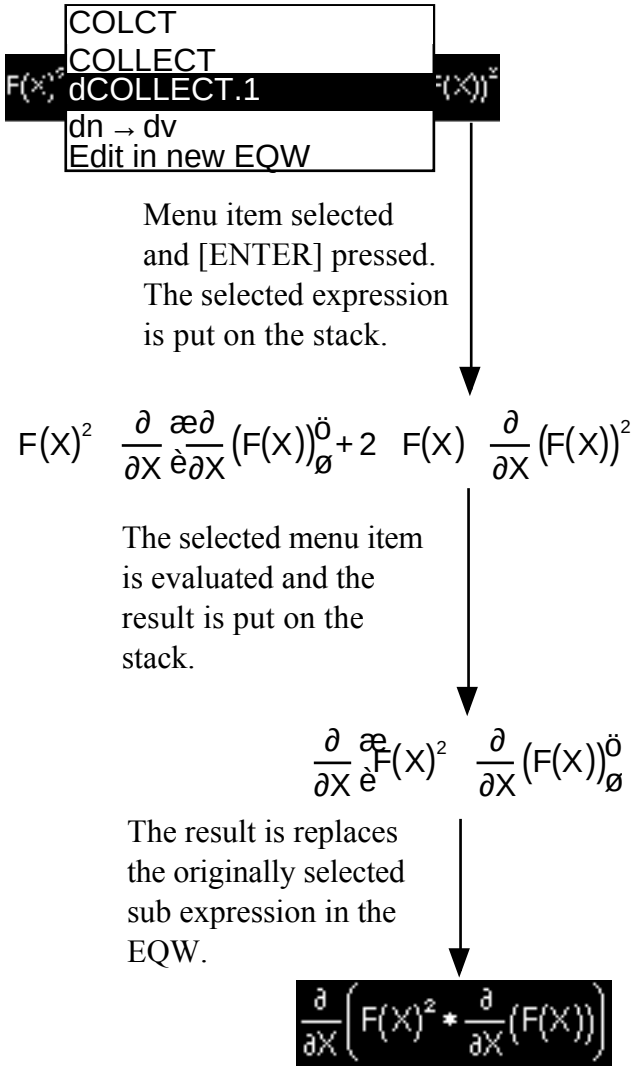


$$\frac{\partial}{\partial X}\left(F(X)^2\right) \quad \frac{\partial}{\partial X}(F(X))^2$$

is displayed inverse in the EQW. Select the sub expression:

$$F(X)^2 \quad \frac{\partial}{\partial X}(F(X))$$

Now the EQW displays:



$$\frac{\partial}{\partial X}\left(F(X)^2 * \frac{\partial}{\partial X}(F(X))\right)$$

Press again CUSTOM, and select again dCOLLECT.1. When the program is ready, the EQW displays:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{3} * F(X)^3\right)\right)$$

Voila! Press ENTER to put the completely collected expression on the stack.

We will use the same technique for the next example that wasn't completely collected, namely the expression:

$$2\ F(X)^2\ \frac{\partial}{\partial X}(F(X))\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right) + 2\ F(X)\ \frac{\partial}{\partial X}(F(X))^3$$

Go to the EQW and enter the above expression. Select the whole expression and use the pop-up like before for running the program dCOLLECT.1. The result is:

$$\frac{\partial}{\partial X}\left(F(X)^2\ \frac{\partial}{\partial X}(F(X))^2\right)$$

Still in the EQW select the sub expression:

$$F(X)^2\ \frac{\partial}{\partial X}(F(X))^2$$

and then use the pop-up again to run dCOLLECT.1. Now the result is:

$$\frac{\partial}{\partial X}\left(\frac{1}{3}\ \frac{\partial}{\partial X}(F(X))^3\ \frac{\partial}{\partial X}(F(X))\right)$$

We can go further and put the factor $\frac{1}{3}$ in one of the two derivatives, getting one of the results:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{3}\right)\ F(X)^3\right)\ \frac{\partial}{\partial X}(F(X))$$

or:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))^3\ \frac{\partial}{\partial X}\left(\frac{1}{3}\right)\ F(X)\right)$$

You have noticed that we put also the menu item Edit in new EQW in the pop-up menu. This is the title of the menu item. When this menu item is selected, then the corresponding object << EQW >> will be executed. The command EQW is the programmable command for starting the EQW. It just needs one algebraic, which it then takes in the EQW for editing. In our case it is not necessary to start a new EQW, but it demonstrates how flexible the HP49G is. In the EQW select the sub expression:

$$\frac{1}{3}\ \frac{\partial}{\partial X}(F(X))^3$$

While the sub expression

$$\frac{1}{3}\ \frac{\partial}{\partial X}(F(X))^3\ \frac{\partial}{\partial X}(F(X))$$

is selected, press ▼ to select $\frac{1}{3}$. Then press → and then ▶ to add

the factor

$$\frac{\partial}{\partial X}\left(F(X)^3\right)$$

to the selected sub expression. Now

$$\frac{1}{3}\ \frac{\partial}{\partial X}\left(F(X)^3\right)$$

is selected. Use the pop-up menu to select Edit in new EQW and run the program << EQW >>. The expression

$$\frac{1}{3}\ \frac{\partial}{\partial X}\left(F(X)^3\right)$$

appears in a new EQW alone, while the other EQW waits suspended in the background. Select the whole expression

$$\frac{1}{3}\ \frac{\partial}{\partial X}\left(F(X)^3\right)$$

in the new EQW and let dCOLLECT.1 run from the popup menu. The result is:

$$\frac{\partial}{\partial X}\left(\frac{1}{3}\ F(X)^3\right)$$

Press ENTER to quit the new EQW, take the algebraic object

$$\frac{\partial}{\partial X}\left(\frac{1}{3}\ F(X)^3\right)$$

and replace the selected object of the old EQW, which was:

$$\frac{1}{3}\ \frac{\partial}{\partial X}\left(F(X)^3\right)$$

Now the old EQW is active again and contains the expression:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{3}\ F(X)^3\right)\right)\ \frac{\partial}{\partial X}\left(F(X)\right)$$

At this point you could press ENTER to put the expression on the stack. But we want to see how the other possible result, namely:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)^3\right)\ \frac{\partial}{\partial X}\left(\frac{1}{3}\ F(X)\right)\right)$$

can be obtained. Press UNDO the last operation and turn the algebraic again to:

$$\frac{\partial}{\partial X}\left(\frac{1}{3}\ \frac{\partial}{\partial X}\left(F(X)^3\right)\ \frac{\partial}{\partial X}\left(F(X)\right)\right)$$

Select the factor $\frac{1}{3}$. Press ⇆ and then ▶ to exchange the positions of $\frac{1}{3}$ and $\frac{\partial}{\partial X}\left(F(X)^3\right)$. Press ⇨ and then ▶ to add $\frac{\partial}{\partial X}\left(F(X)\right)$ to the selection. Now use dCOLLECT.1 from the pop-up menu to convert the selected sub expression:

$$\frac{1}{3}\ \frac{\partial}{\partial X}\left(F(X)\right)$$

to:

$$\frac{\partial}{\partial X}\left(\frac{1}{3}\right)\ F(X)$$

Now the EQW contains:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)^3\right)\right)\ \frac{\partial}{\partial X}\left(\frac{1}{3}\right)\ F(X)$$

After these exciting interactive manoeuvres, let's see why dCOLLECT.1 can't return the completely collected result. The program took the expression:

$$2\ F(X)^2\ \frac{\partial}{\partial X}(F(X))\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)+2\ F(X)\ \frac{\partial}{\partial X}(F(X))^3$$

and converted it to the two terms:

$$2\ F(X)^2\ \frac{\partial}{\partial X}(F(X))\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)$$

and:

$$2\ F(X)\ \frac{\partial}{\partial X}(F(X))^3$$

The first term was converted to:

$$F(X)^2\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))^2\right)$$

using the rule of differentiation of powers. The second was converted to:

$$\frac{\partial}{\partial X}\left((F(X))^2\right)\ \frac{\partial}{\partial X}(F(X))^2$$

using the same rule. The sum of the two terms,

$$F(X)^2\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))^2\right)+\frac{\partial}{\partial X}\left((F(X))^2\right)\ \frac{\partial}{\partial X}(F(X))^2$$

was converted to:

$$\frac{\partial}{\partial X}\left(F(X)^2\ \frac{\partial}{\partial X}(F(X))^2\right)$$

using the rule of differentiation of products. After this, the program didn't check if application of this rule, namely:

$f(x)\ \frac{\partial g(x)}{\partial x}+g(x)\ \frac{\partial f(x)}{\partial x}=\frac{\partial f(x)\ g(x)}{\partial x}$, creates a product $f(x)\ g(x)$, in our case $F(X)^2\ \frac{\partial}{\partial X}(F(X))^2$, which itself can be further collected using the rule of differentiation of powers.

We do the same for the example

$$(F(X)+1)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)+\frac{\partial}{\partial X}(F(X))^2$$

Enter this in the EQW, select the whole expression and use the pop-up to run dCOLLECT.1 and get:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\ F(X)\right)+\frac{\partial}{\partial X}(F(X))$$

Still in the EQW you can either select the sub expression:

$$\frac{\partial}{\partial X}(F(X))\ F(X)$$

and run dCOLLECT.1 to transform the sub expression to:

$$\frac{\partial}{\partial X}\left(\frac{1}{2}\ F(X)^2\right)$$

thus obtaining the result:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{2}\right)\ F(X)^2\right)+\frac{\partial}{\partial X}(F(X))$$

Or you can select the sub expression:

$$\frac{\partial}{\partial X}(F(X))\ F(X)+\frac{\partial}{\partial X}(F(X))$$

and use the pop-up menu to evaluate COLLECT and transform the sub expression to:

$$(F(X)+1)\ \frac{\partial}{\partial X}(F(X))$$

thus gaining the result:

$$\frac{\partial}{\partial X}\left((F(X)+1)\ \frac{\partial}{\partial X}(F(X))\right)$$

In this example the program dCOLLECT.1 transformed the expression:

$$(F(X)+1)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)+\frac{\partial}{\partial X}(F(X))^2$$

to its terms,

$$\frac{\partial}{\partial X}(F(X))^2\ ,$$

$$F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right),$$

and $1\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)$

The first term was transformed to:

$$\frac{\partial}{\partial X}(F(X))\ \frac{\partial}{\partial X}(F(X))$$

Then the sum of the first and the second terms,

$$\frac{\partial}{\partial X}(F(X))\ \frac{\partial}{\partial X}(F(X))+F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)$$

was transformed to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\ F(X)\right)$$

using the rule of differentiation of products. Then the result

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\ F(X)\right)$$

replaced the two terms:

$$\frac{\partial}{\partial X}\left(F(X)\right)^2$$

and:

$$F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right)$$

in the list of terms. At this point, like in the previous example, the program didn't check if the result of application of the rule

$$f(x)\ \frac{\partial g(x)}{\partial x} + g(x)\ \frac{\partial f(x)}{\partial x} = \frac{\partial f(x)\ g(x)}{\partial x}$$

created a product $f(x)\ g(x)$, in our case

$$\frac{\partial}{\partial X}\left(F(X)\right)\ F(X)\ )$$

which itself can be further collected to

$$\frac{\partial}{\partial X}\left(\frac{1}{2}\ F(X)^2\right)$$

using the rule of differentiation of powers. Instead of doing this, the program constructed the sum of the new partially collected term

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\ F(X)\right)$$

and of the term

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right)$$

and obtained the sum:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\ F(X)\right) + \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right)$$

It used the rule of differentiation of sums to convert this result to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\ F(X)\right) + \frac{\partial}{\partial X}\left(F(X)\right)$$

which was what it returned.

Then we had the example:

$$F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right) + \frac{\partial}{\partial X}\left(F(X)\right)^2$$

for which dCOLLECT.1 returned:

$$\frac{\partial}{\partial X}\left(F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)\right)$$

but not

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{2}\ F(X)^2\right)\right)$$

Let's see what the program did. It first converted the expression to its terms,

$$F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)^2\right)$$

and $2\ F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)$

Then it tried to apply the rule of differentiation of powers for each of these terms. Doing that it converted the second term to:

$$\frac{\partial}{\partial X}\left(F(X)\right)\ \frac{\partial}{\partial X}\left(F(X)\right)$$

though this is not collection of differential forms. Then it started checking all possible pair wise sums of all terms for application of the rule of differentiation of products, sums, etc. When it checked the sum of the first and the second term:

$$F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)^2\right)+\frac{\partial}{\partial X}\left(F(X)\right)\ \frac{\partial}{\partial X}\left(F(X)\right)$$

it saw that this is convertible to:

$$\frac{\partial}{\partial X}\left(F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)^2\right)$$

using the rule of product differentiation. And so it converted it. (Or else why the trouble? ;-)) Then it removed the two terms from which the collected term came from and kept the new collected term, which was also the only one that remained. At this point the program didn't bother to check if the new expression:

$$F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)$$

inside the parentheses can be converted to:

$$\frac{\partial}{\partial X}\left(\frac{1}{2}\ F(X)^2\right)$$

and so it returned the result:

$$\frac{\partial}{\partial X}\left(F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)^2\right)$$

Of course you can again use the EQW to select

$$F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)$$

and convert it to:

$$\frac{\partial}{\partial X}\left(\frac{1}{2}\ F(X)^2\right)$$

using dCOLLECT.1 from the pop-up menu of STARTEQW, gaining thus the completely collected result:

$$\frac{\partial}{\partial X}\left(\left(\frac{\partial}{\partial X}\left(\frac{1}{2}\ F(X)^2\right)\right)^2\right)$$

Let's now get a closer look at the workings of the program. Doing this we are going to see some very interesting behaviour patterns of the HP49G too. First of all the program uses PUSH to store the current settings of the user, because it males changes to flag settings and we don't want a program to change the modes of the calculator behind the back of the user. Then the program calls dn → dv and → TERMS. This results in a list of all terms of an expression. Notice that dn → dv replaces all occurrences of derivatives in the d1 notation to derivatives in the $\frac{\partial}{\partial}$ notation. The program → TERMS uses the command

FDISTRIB. Fortunately FDISTRIB does completely distribution of and / over + and −, but it expand otherwise and so *it doesn't change the derivatives of the expression back to* d1 *notation*. That means, if you enter for example

$$\frac{\partial}{\partial X}(F(X))\ (X+3)$$

and press FDISTRIB, then you will get:

$$X\ \frac{\partial}{\partial X}(F(X))+3\ \frac{\partial}{\partial X}(F(X))$$

This is very good because we can build up the list of all terms of an arbitrary expression without having to care if some special syntax or notation will be destroyed. Why do we want a list of all terms of the original expression? Why don't we work with the expression itself? Well, imagine how many possibilities there are, to built up arbitrary expressions. We can use have them in an endless variety, and so finding general patterns for matching, would simply become impossible. Chopping the expression in a list of all terms, we can be sure that each one of the terms is a product. When we can ensure that some given expression is of a certain type, then we narrow the variety of what must be done, to convert the expression according to our needs. But, of course, we have to do what has to be done, for each of the terms. Having the terms in a list, we use a loop to apply the first of the converting procedures to each term.

Each one of the terms gets converted to a list of its factors using the command FACTORS. First of all, we can use this command safely because it also doesn't change the derivatives and leaves them in our $\frac{\partial}{\partial}$ notation. Why do we want the factors of each term? Well, the answer has to do with the fact that we apply the rule for derivation of powers first. Simply using pattern matching here would make our life very difficult. Imagine for example that we have the term:

$$F(X)\ \frac{\partial}{\partial X}(F(X))$$

Here we can use the pattern matching with the pattern:

$$\begin{matrix}\grave{\imath}\\ \acute{\imath}\\ \hat{\imath}\end{matrix}\&A\ \frac{\partial}{\partial \&V}(\&A)\ \ \frac{\partial}{\partial \&V}\begin{matrix}\ae\\ \c{c}\\ \grave{e}\end{matrix}\frac{\&A^2}{2}\begin{matrix}\ddot{o}\ddot{u}\\ \acute{y}\\ \o p\end{matrix}$$

But this is one of the many cases where the rule of differentiation of powers can be used to collect differential forms. What would be for example with:

$$F(X)\ \frac{\partial}{\partial X}\left(F(X)^2\right)$$

or:

$$F(X)^2\ \frac{\partial}{\partial X}(F(X))$$

or:

$$F(X)^2\ \frac{\partial}{\partial X}(F(X))^2$$

or…? Obviously we must find a more general method. And to do that we do a bit mathematics first. Suppose that we have the derivative:

$$\frac{\partial}{\partial X}\left(F(X)^n\right)=n\ F(X)^{n-1}\ \frac{\partial}{\partial X}(F(X))$$

This already shows that:

$$F(X)^{n-1}\ \frac{\partial}{\partial X}(F(X))=\frac{1}{n}\ \frac{\partial}{\partial X}\left(F(X)^n\right)$$

That means for us, that if we have some expression which contains the pattern

$$F(X)^n \ \frac{\partial}{\partial X}\left(F(X)\right)$$

with arbitrary n, we can replace it with

$$\frac{1}{n+1} \ \frac{\partial}{\partial X}\left(F(X)^{n+1}\right)$$

This is more general, but not as general as we wish. Because we might also have the pattern

$$F(X)^n \ \frac{\partial}{\partial X}\left(F(X)^m\right)$$

in an expression, where n and m are also arbitrary. In this case we have:

$$F(X)^n \ \frac{\partial}{\partial X}\left(F(X)^m\right) = F(X)^n \ m \ F(X)^{m-1} \ \frac{\partial}{\partial X}\left(F(X)\right)$$

$$F(X)^n \ \frac{\partial}{\partial X}\left(F(X)^m\right) = m \ F(X)^{n+m-1} \ \frac{\partial}{\partial X}\left(F(X)\right)$$

$$F(X)^n \ \frac{\partial}{\partial X}\left(F(X)^m\right) = \frac{m}{n+m} \ \frac{\partial}{\partial X}\left(F(X)^{n+m}\right)$$

and thus we must replace

$$F(X)^n \ \frac{\partial}{\partial X}\left(F(X)^m\right)$$

with

$$\frac{m}{n+m} \ \frac{\partial}{\partial X}\left(F(X)^{n+m}\right)$$

To understand this better, expand

$$F(X)^n \ \frac{\partial}{\partial X}\left(F(X)^m\right)$$

and

$$\frac{m}{n+m} \ \frac{\partial}{\partial X}\left(F(X)^{n+m}\right)$$

The two results are equal. But this still isn't general enough. (We are very demanding, aren't we? ;-)) We could also have some term that contains

$$F(X)^n \ \frac{\partial}{\partial X}\left(F(X)^m\right)^p$$

What to do in such cases? Well, one of the possibilities that we have, is to convert

$$F(X)^n \ \frac{\partial}{\partial X}\left(F(X)^m\right)^p$$

to

$$F(X)^n \ \frac{\partial}{\partial X}\left(F(X)^m\right) \ \frac{\partial}{\partial X}\left(F(X)^m\right)^{p-1}$$

and then consider the sub product

$$F(X)^n \ \frac{\partial}{\partial X}\left(F(X)^m\right)$$

We can do collection to:

$$\frac{m}{n+m}\ \frac{\partial}{\partial X}\left(F(X)^{n+m}\right)$$

and then multiply by:

$$\frac{\partial}{\partial X}\left(F(X)^{m}\right)^{p-1}$$

getting the result:

$$F(X)^{n}\ \ \frac{\partial}{\partial X}\left(F(X)^{m}\right)\ \ \frac{\partial}{\partial X}\left(F(X)^{m}\right)^{p-1}$$

This is what how the rule of differentiation of powers is implemented in the program for collection of differential forms. First of all, the list of factors and multiplicities (factor power) is converted to a list of lists. Each of the sub lists contains the factor and its power. Right after this, all factors are separated. If they contain any derivative then they are added in the list of differential factors. Otherwise they are added in the list of (normal) factors. At this point we consider the fact that some terms will eventually contain more than one differential factors. Take for example

$$\frac{\partial}{\partial X}\left(F(X)\right)\ \ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right)$$

which contains two factors that are derivatives. In such cases the program in its current incarnation prefers the highest derivative. (You might started guessing that many incarnations will follow, what Bhuvanesh? ;-)) The program leaves the highest derivative as the only expression in the list of differential factors, and moves all the rest into the list of the other factors. We can easily decide which one is the highest derivative in the list of differential factors, because the command FACTORS returns always the highest derivative as the first element in the list of factors and multiplicities, while the lower derivatives follow in order.

Then the program converts the differential factors list:

$$\{\text{highestDerivative}\ \ \text{power}\}$$

to the differential factors list:

$$\{\text{highestDerivative}\ \ \text{highestDerivative}\ \ \text{power}-1\}$$

This corresponds to the step:

$$\frac{\partial}{\partial X}\left(F(X)^{m}\right)^{p} = \frac{\partial}{\partial X}\left(F(X)^{m}\right)\ \ \frac{\partial}{\partial X}\left(F(X)^{m}\right)^{p-1}.$$

After this, the program checks to find if the expression inside the parentheses of $\frac{\partial}{\partial X}(\ )$ in the highest derivative, occurs also in the list of the other factors. If it does, then it puts the factor

$$\frac{\partial}{\partial X}\left(F(X)^{n+m}\right)$$

into the first position of the list of differential factors:

$$\{\text{highestDerivative}\ \ \text{highestDerivative}\ \ \text{power}-1\}$$

and adds the list:

$$\{F(X)\ \ -(n+m-1)\ \ n+m\ \ -1\}$$

to the list of other factors, which is equivalent to adding the factors $F(X)^{-(n+m-1)}$ and $(n+m)^{-1}$ to the list of the other factors.

We make an example for getting the idea of the whole procedure up to now. Suppose that we have:

$$F(X)^2 \quad \frac{\partial}{\partial X}\left(F(X)^5\right)$$

This will be converted to:

$$5 \quad F(X)^6 \quad \frac{\partial}{\partial X}\left(F(X)\right)$$

by $dn \rightarrow dv$. After this the program $\rightarrow$ TERMS converts that to:

$$\begin{bmatrix} 5 & F(X)^6 & \frac{\partial}{\partial X}\left(F(X)\right) \end{bmatrix}$$

The command FACTORS converts the term

$$5 \quad F(X)^6 \quad \frac{\partial}{\partial X}\left(F(X)\right)$$

to the list

$$\begin{bmatrix} 5 & 1. & \frac{\partial}{\partial X}\left(F(X)\right) & 1. & F(X) & 6. \end{bmatrix}$$

Our program then converts this list to:

$$\begin{bmatrix} \{5 \quad 1.\} & \begin{bmatrix} \frac{\partial}{\partial X}\left(F(X)\right) & 1. \end{bmatrix} & \{F(X) \quad 6.\} \end{bmatrix}$$

and separates the list to

$$\{5 \quad 1. \quad F(X) \quad 6.\}$$

and

$$\begin{bmatrix} \begin{bmatrix} \frac{\partial}{\partial X}\left(F(X)\right) & 1. \end{bmatrix} \end{bmatrix}$$

The list of the differential factors is converted to:

$$\begin{bmatrix} \begin{bmatrix} \frac{\partial}{\partial X}\left(F(X)\right) & \frac{\partial}{\partial X}\left(F(X)\right) & 0. \end{bmatrix} \end{bmatrix}$$

Then the factor

$$\frac{\partial}{\partial X}\left(F(X)^7\right)$$

is put into the first position of the list of differential factors, turning it to:

$$\begin{bmatrix} \begin{bmatrix} \frac{\partial}{\partial X}\left(F(X)^7\right) & \frac{\partial}{\partial X}\left(F(X)\right) & 0. \end{bmatrix} \end{bmatrix}$$

After this the list

$$\{F(X) \quad -6. \quad 7 \quad -1.\}$$

is added to the list of other factors, turning it to

$$\{5 \quad 1. \quad F(X) \quad 6. \quad F(X) \quad -6. \quad 7 \quad -1.\}$$

Then the list of factors is turned to

$$\begin{bmatrix} 5 & F(X)^6 & F(X)^{-6} & \frac{1}{7} \end{bmatrix}$$

The product of the list is built up using ΠLIST and EXPAND, which returns $\frac{5}{7}$.

We proceed describing the program. Because the product of normal factors might also contain derivatives (the lower derivatives which, if they exist, are put in the list of normal factors), and because we used EXPAND, which turns these derivatives to d1 notation, we call the program dn → dv once again to convert back to $\frac{\partial}{\partial}$ notation. Then we turn the list of differential factors to the product:

$$\frac{\partial}{\partial X}\left(F(X)^{n+m}\right)\ \frac{\partial}{\partial X}\left(F(X)\right)^{p-1}$$

which for the above example translates to:

$$\frac{\partial}{\partial X}\left(F(X)^{7}\right)\ \frac{\partial}{\partial X}\left(F(X)\right)^{0}$$

or simply:

$$\frac{\partial}{\partial X}\left(F(X)^{7}\right)$$

(Note that we don't need to expand in order to simplify our example, because the operations are done with RPL syntax and so the sequence:

$$\frac{\partial}{\partial X}\left(F(X)\right)\ \ 0\ \ \wedge$$

automatically returns 1. Also,

$$\frac{\partial}{\partial X}\left(F(X)^{7}\right)\ \ 1$$

returns automatically $\frac{\partial}{\partial X}\left(F(X)^{7}\right)$.)

We make another example in abbreviated form of the inner workings of the program up to this point, using the expression:

$$\frac{\partial}{\partial X}\left(F(X)^{2}\right)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)^{3}\right)^{2}$$

$$\frac{\partial}{\partial X}\left(F(X)^{2}\right)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)^{3}\right)^{2}\ \xrightarrow{\text{dn->dv}}$$

$$18\ F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)^{5}\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)^{3}\right)^{2}\ \xrightarrow{\text{->TERMS}}$$

$$\{\ 18\ F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)^{5}\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)^{3}\right)^{2}\ \}\ \xrightarrow{\text{FACTORS}}$$

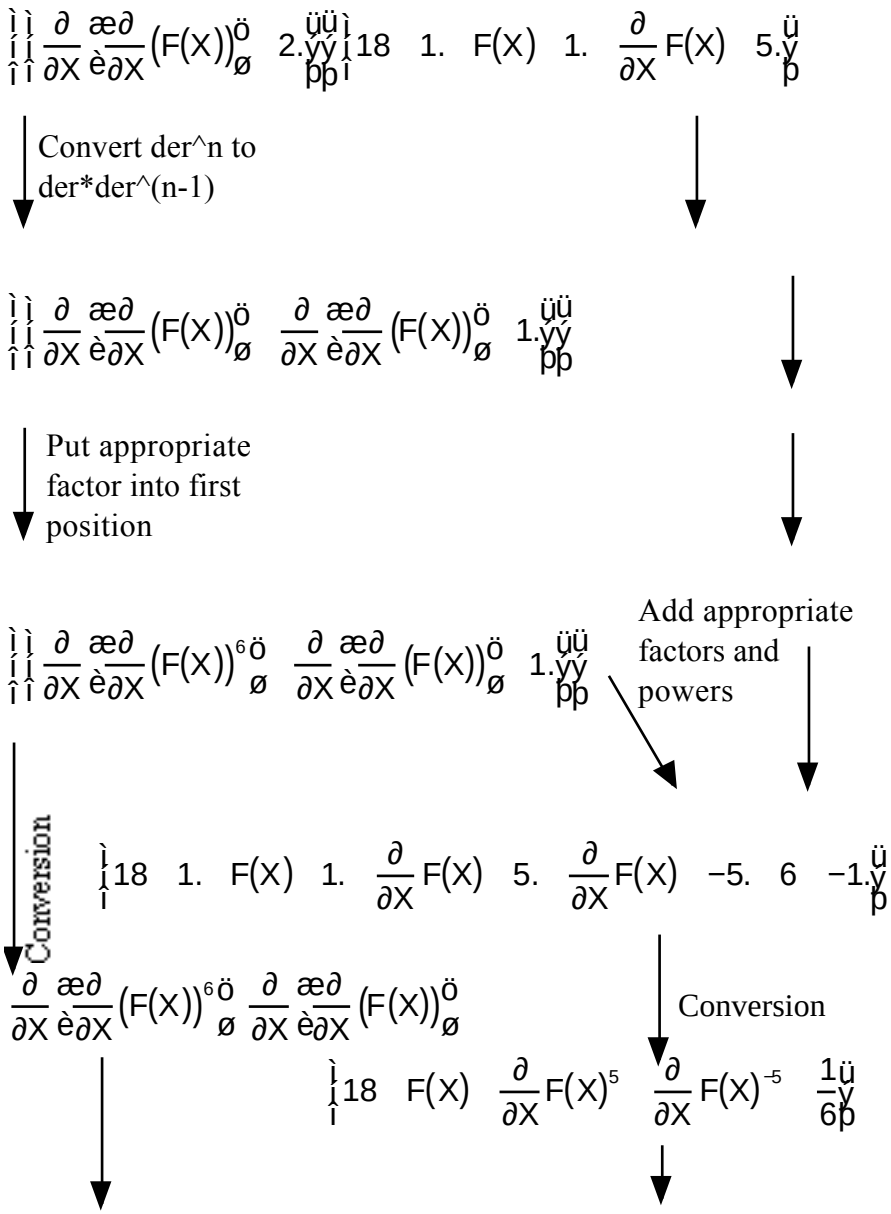
$$\{\ 18\ \ 1.\ \ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)^{3}\right)\ \ 2.\ \ \frac{\partial}{\partial X}\left(F(X)\right)\ \ 5.\ \ F(X)\ \ 1.\ \}\ \xrightarrow{\text{Conversion to list of lists}}$$

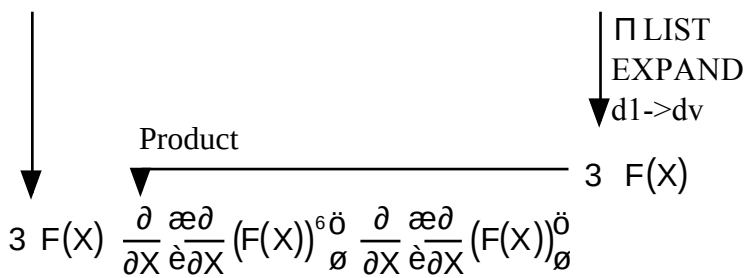
$$\{\ \{18\ \ 1.\}\ \ \{\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)^{3}\right)\ \ 2.\ \}\ \ \{\ \frac{\partial}{\partial X}\left(F(X)\right)\ \ 5.\ \}\ \ \{F(X)\ \ 1.\}\ \}$$

Separation of factors

$$\{\ \{\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)^{3}\right)\ \ 2.\ \}\ \ \{\ \frac{\partial}{\partial X}\left(F(X)\right)\ \ 5.\ \}\ \ \{18\ \ 1.\ \ F(X)\ \ 1.\}\ \}$$

Hold only highest derivative





One question that you might ask is, why do we prefer the highest derivative? Why not the lowest? Well, that presumably has to do with the fact that the things I work with, demand quite often to do that. But if it fits your needs better, you can of course tell me to change that. Just mail me your wishes and I'll see what can be done. The perfect thing would be of course to check all derivatives and select that particular one which best fits for collection of differential forms according to the rule of differentiation of power. (Yet another future incarnation? Or rather in*bit*ation;-))

Right after this rather complex part that deals with the rule of differentiation of powers, we have a somewhat simpler part, which tries to apply the rule of differentiation of products. It does this by using pattern matching twice. The whole pattern matching is inside a CASE−THEN−END clause. If one match works, then the subsequent matches are not performed at all. Why do we do that this way? Couldn't we just keep on doing pattern matching until nothing changes? To answer this we must take a look at the matching patterns at this point. In general, we will convert expressions of type:

$$F(X)\ \frac{\partial G(X)}{\partial X} + G(X)\ \frac{\partial F(X)}{\partial X}$$

to expressions of type:

$$\frac{\partial F(X)\ G(X)}{\partial X}$$

We see that this match doesn't create a sum, that would eventually be of exactly the same type. Because we do pattern matching to all possible sums of two terms of the original expression, no new sum can be created by pattern matching, so that no subsequent matching of the same type is necessary. This fact, together with the fact that the used command ↑ MATCH does pattern matching starting at the most inner nested sub expressions and continues matching until the top expressions are reached, makes a single pattern matching operation sufficient. First we try to match:

$$F(X)\ \frac{\partial G(X)}{\partial X} + G(X)\ \frac{\partial F(X)}{\partial X}$$

to:

$$\frac{\partial F(X)\ G(X)}{\partial X}$$

If it works, we leave the CASE−THEN−END clause. If it doesn't, we try to match:

$$F(X)\ \frac{\partial G(X)}{\partial X} + \frac{\partial F(X)}{\partial X}\ G(X)$$

to:

$$\frac{\partial F(X)\ G(X)}{\partial X}$$

If it works, we leave the clause. If it doesn't, then we collect and repeat the two above pattern matching operations once again. We do that because our expression might also be in the form:

$$A\ F(X)\ \frac{\partial G(X)}{\partial X} + A\ G(X)\ \frac{\partial F(X)}{\partial X}$$

or similar. So we collect to convert it to:

$$A\ \left(F(X)\ \frac{\partial G(X)}{\partial X} + G(X)\ \frac{\partial F(X)}{\partial X}\right)$$

so that the hidden pattern:

$$F(X)\ \frac{\partial G(X)}{\partial X} + G(X)\ \frac{\partial F(X)}{\partial X}$$

becomes visible for ↑ MATCH. Note that COLLECT will not destroy the $\frac{\partial}{\partial}$ notation of our expressions. *Also note that COLLECT will not collect expressions inside the parentheses of* $\frac{\partial}{\partial}(\ )$. It only collects "outside".

After this, the program uses pattern matching to apply the rule of differentiation for expressions multiplied with some constant. At this step we try to convert all expressions of type:

$$\text{constant}\ \frac{\partial F(X)}{\partial X}$$

to expressions of type:

$$\frac{\partial \text{constant}\ F(X)}{\partial X},$$

only if the expression constant is really a constant, i.e. only if it doesn't contain the variable of derivation. We use the program POSNAME and the method described on pages 2-2 to 2-3.

The next step is to apply the rule of differentiation of sums, again by doing pattern matching. We match expressions of the form:

$$\frac{\partial F(X)}{\partial X} + \frac{\partial G(X)}{\partial X}$$

to:

$$\frac{\partial F(X) + G(X)}{\partial X}$$

and expressions of the form:

$$\frac{\partial F(X)}{\partial X} - \frac{\partial G(X)}{\partial X}$$

to:

$$\frac{\partial F(X) - G(X)}{\partial X}$$

During all the above operations we kept track of the success of the application of the rules. After all rules are applied, we check if some of them was successful. If so, then we remove the two terms of which the currently examined sum consists from the list of terms of the original expressions. Then we put the result of the pattern matching operations in the same list. This makes the list shrink by one element. Then we start over building up all possible sums of two terms until all sums have been examined.

This is in brief the way dCOLLECT.1 works. There are some additional things that are interesting. We are going to take a look at them now. First of all, we have seen that there are commands, like FACTORS or COLLECT, which retain our $\frac{\partial}{\partial}$ notation of derivatives. FACTOR also belongs to these commands, but EXPAND doesn't belong to them, as it will convert all formal derivatives to d1 notation. In cases where the specific notation of derivatives is of importance, you should always check to see if some command that you want to use retains the notation, or converts it to d1.

We also saw that FACTORS will return the highest derivatives in positions before the lower derivatives in the list of factors and multiplicities. But this works only for "simple" expressions. For example, if you enter:

$$\frac{\partial}{\partial X}(F(X))\ F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)$$

and press FACTORS, then you get the result:

$$\left\{\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)\ 1.\ \frac{\partial}{\partial X}(F(X))\ 1.\ F(X)\ 1.\right\}$$

But if you enter:

$$\frac{\partial}{\partial X}(F(X))\ F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)+\frac{\partial}{\partial X}(F(X))\ F(X)$$

and press FACTORS, then the HP49G will return the result:

$$\left\{\frac{\partial}{\partial X}(F(X))\ 1.\ F(X)\ 1.\ \frac{\partial}{\partial X}(F(X))+1\ 1.\right\}.$$

This is an additional reason for chopping our expressions to its terms, so that we can be sure that when we use FACTORS, there will not be any factors that are themselves sums, and so the highest derivative will be the first element in the list of factors. FACTOR works in a similar way. If you enter:

$$\frac{\partial}{\partial X}(F(X))\ F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)$$

and press FACTOR, then the result is:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right) \quad \frac{\partial}{\partial X}(F(X)) \quad F(X)\ ,$$

in which again the derivatives appear in order higher to lower. If you enter:

$$\frac{\partial}{\partial X}(F(X)) \quad F(X) \quad \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right) + \frac{\partial}{\partial X}(F(X)) \quad F(X)$$

and press FACTOR, then you get:

$$\frac{\partial}{\partial X}(F(X)) \quad F(X) \quad \left(\frac{\partial}{\partial X}\frac{\partial}{\partial X}(F(X)) + 1\right)\ ,$$

in which the factors appear in the same order like in the result of the command FACTORS. COLLECT returns a result with the exactly opposite order. Enter:

$$\frac{\partial}{\partial X}(F(X)) \quad F(X) \quad \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)$$

and press COLLECT to get:

$$F(X) \quad \frac{\partial}{\partial X}(F(X)) \quad \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)$$

Or enter:

$$\frac{\partial}{\partial X}(F(X)) \quad F(X) \quad \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right) + \frac{\partial}{\partial X}(F(X)) \quad F(X)$$

and press COLLECT to get:

$$\left(\frac{\partial}{\partial X}\frac{\partial}{\partial X}(F(X)) + 1\right) \quad F(X) \quad \frac{\partial}{\partial X}(F(X))$$

EXPAND has also its distinct idiosyncrasy. Enter:

$$\frac{\partial}{\partial X}(F(X)) \quad F(X) \quad \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)$$

and press EXPAND. The result is $F(X)\ d1F(X)\ d1d1F(X)$, which shows that in products the command EXPAND puts the derivatives in the order lower to higher. Enter:

$$\frac{\partial}{\partial X}(F(X)) \quad F(X) \quad \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right) + \frac{\partial}{\partial X}(F(X)) \quad F(X)$$

and expand again to get $F(X)\ d1F(X)\ d1d1F(X) + F(X)\ d1F(X)$, which shows that the terms are ordered in a way, that terms containing the highest derivative appear first, followed by terms with lower derivatives. The order of the derivatives inside each term is again lower to higher.

Another interesting thing that we see in the program dCOLLECT.1, is the technique of breaking out of FOR loops by storing a value in the counter, that exceeds the upper limit of iteration. Consider for the code:

```
<<
  1 10
  FOR I
    IF
      I 5 ≥
    THEN
      100 'I' STO
    END
  NEXT
>>
```

In this code, the loop will run until the iteration variable I has the value 5. At this point, a value of 100 is stored in I, which makes the program leave the loop when the NEXT command is executed, since the value of 100 exceeds the upper iteration limit which is 10. The iteration variable is a local variable that exists only inside the loop. As long as we are inside the loop, we can not only use its current value, but also store new values in it using STO, STO+ and so on. So we have a way to exit FOR loops in a polite and civilised way, by simply adding the upper limit of iteration to the iteration variable. We can check if something particular happens, that makes us wanting to leave the loop, and if it happens we can use STO+ to make the iteration variable greater than the upper limit of iteration. Note however, that the loop isn't exited automatically when we store some value in the iteration variable. All the subsequent commands until NEXT will be executed. Only when the command NEXT is executed does the HP49G check the current value of the iteration variable, and decides to leave the loop if it exceeds the upper limit of iteration. It is not an emergency right now without any questions exit. But even such an exit can be implemented. You just have to put all the commands that shouldn't be executed when the iteration variable exceeds the upper limit, into an IF−THEN−END clause. Consider the code:

```
<<
  DUP SIZE
   → ourList up
  <<
    1 up
    FOR I
      ourList I GET SQ
      IF DUP 1000 ≥
      THEN up 'I' STO+
      END
      IF I up ≤
      THEN 1 + 3 ^
      END
    NEXT
>>
```

This program takes a list of numbers as argument, and starts a loop in which it calculates the square of the square of the number. If this square is greater than or equal to 1000, the program adds the upper limit of iteration to the iteration variable. Then it checks if the iteration variable exceeds the upper limit iteration. If it doesn't, then it adds 1 to the square of the number and raises the result to the third power. But if it does, then it does nothing more and continues after the END of the second IF−THEN−END clause. Since the next command is NEXT, the current value of the iteration variable is checked. Since it is greater than the upper limit, the loop is exited without performing the code `1 + 3 ^`, immediately.

Perhaps you have asked yourself, why do we add {1 1} to the list of factors before we use ΠLIST? And why do we add 0 to the list of terms before we use ΣLIST? Well, we do that to avoid the error "Invalid dimension, in case the lists contain only one element or no elements at all. In our case, instead of checking how many elements some list has, we can add {1 1} to the list before we use ΠLIST, to make sure that there will be a result in any case. If the list has only a single element, then multiplying it with 1 will not cause any trouble. If it has no elements at all, then we find the factor 1, which also doesn't change our expression. The same considerations apply to adding 0 or {0 0} to a list before we use ΣLIST. Note however that this technique is applicable to our case and to other cases, but there can be cases where we shouldn't use it. If for example some result does depend on the fact that a list has only one element, then we should better check the size of the list.

Talking about lists, we shouldn't forget another important feature of the command SUB. This command can be used to build up sub parts of many objects. When it is used with lists, it takes a list from stack level 3, the starting position from stack level 2, and the end position from stack level 1. It returns a new list which contains the elements of the old list, starting and ending at the specified positions. For example, entering {1 2 3 4 5}, 2, 4, and pressing SUB, returns {2 3 4}. But this command has also some very convenient properties. Enter again {1 2 3 4 5}, then 2, and then 10, and press SUB. Though the

list has only 5 elements, the HP49G doesn't error out, but simply returns {2 3 4 5}. Enter once more {1 2 3 4 5}, then enter −3 (!) and 3, and press SUB to get {1 2 3}. And it gets even better. enter {1 2 3 4 5}, then 10, and then 15, and press SUB to get an empty list as result. There is no "out of range" error! (However, if you enter {1 2 3 4 5}, 5, 3, and then press SUB you don't get {3 4 5}, but an empty list as result.)

Enough peculiarities and nice properties. Time to return to our main path again. If the fans of program reinbitation start feeling euphoric, then this must be partly because we are about to think how the program dCOLLECT.1 could get better. Let's first remember why sometimes the program doesn't do complete collection of differential forms. When some applied rule is successful in combining differential forms, it could return a new differential form, which contains an expression between the parentheses of $\frac{\partial}{\partial}(\ )$, which itself can be collected. One possible way to do that would be, to keep on doing pattern matching until nothing more changes. But we can't use this method for collection of differential forms using the rule of differentiation of powers, because especially for this purpose we don't work with pattern matching. Let's work out a method for this case, that is able to continue collecting differential forms into the depths of the algebraic expression. Imagine some hypothetical code, that is able to detect the presence of differential forms, extract the expression in the parentheses of $\frac{\partial}{\partial}(\ )$, and give this expression to dCOLLECT.1. If our code can do that for any nested level of derivatives, then we are able to apply dCOLLECT.1 repeatedly, until nothing changes. Consider the following code, which I will comment after its listing:

```
<<
  DO
    "∂ → DERIV" 3. DISP  @Message
  DO                     @Convert all ∂ to DERIV
    { '∂&V(&A)'
      'DERIV(&A,&V' }
    ↑MATCH
  UNTIL
    NOT
  END
  →LST                   @Convert alg. to list
  'DERIV(A,B)'           @Dummy
  →LST                   @Convert it to list
  DUP 3 GET              @Get the object 'DERIV'
  SWAP 5 GET             @and the invisible APPLY
  → deralg apl           @Store in locals
  <<
    "Object" 3. DISP     @Message
    1
    <<                   @DOSUBS procedure
      NSUB R→I " of " +
      ENDSUB R→I +
      4. DISP                @Message
      CASE
           DUP deralg SAME   @If we have a 'DERIV'
        THEN                 @then do nothing
        END
           DUP apl SAME      @If we have APPLY
        THEN                 @then convert the function
           ROT 1 →ALG        @that is applied to alg.
           UNROT EVAL        @and evaluate APPLY
           IF                @If
              DUP OBJ→       @we have DERIV(expression)
              {DERIV} HEAD
              SAME
           THEN              @then
              DROP2          @apply dCOLLECT to expr.
              dCOLLECT       @(dCOLLECT is new! Its
              OVER OBJ→      @description comes on
              3 DROPN        @page 2-31)
              SWAP 2 →LIST   @and match expression
              ↑MATCH DROP    @with matched expression.
           ELSE
              DROPN
```

```
          END
        END
          EVAL              @In all other cases EVAL
      END
    >>
    DOSUBS
    "DERIV → ∂" 3. DISP     @Message
    HEAD
    DO                      @Convert all DERIV to ∂
      { 'DERIV(&A,&V'
        '∂&V(&A)'} ↑MATCH
    UNTIL
      NOT
    END
  >>
  UNTIL
    SWAP OVER SAME       @Do until nothing changes
  END
>>
```

Let's take a closer look to what the program does. First of all we convert all patterns $\frac{\partial}{\partial X}(F)$ to $\text{DERIV}(F,X)$. Why do we do that? Well, we want later on to transform the algebraic object to an RPL list, using the command list. The problem is that nested derivatives are not exploded to the objects of which they consist. For example, if you enter:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(X^2\right)\right)$$

and press →LST, then the result is the list:

$$\{X\ \ \frac{\partial}{\partial X}\left(X^2\right)\ \ \partial\}$$

and not $\{X\ \ X\ \ X\ \ 2\ \ \hat{}\ \ \partial\ \ \partial\}$, which would be the complete RPL decomposition of the algebraic object:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(X^2\right)\right)$$

But if you enter the same algebraic object using DERIV, that is $\text{DERIV}(\text{DERIV}(X^2,X),X)$, and press →LST, then the result is $\{X\ \ 2\ \ \hat{}\ \ X\ \ \text{'DERIV'}\ \ \#2d\ \ X\ \ \text{'DERIV'}\ \ \#2d\}$. This list is a complete decomposition. It contains two invisible items. If you press OBJ→ to explode the list, then stack levels 2 and 6 seem to contain thin air. If we represent these invisible objects with •, then the list in reality looks like:

$$\{X\ \ 2\ \ \hat{}\ \ X\ \ \text{'DERIV'}\ \ \#2d\ \ \bullet\ \ X\ \ \text{'DERIV'}\ \ \#2d\ \ \bullet\}$$

The strange invisible object • is the command FCNAPPLY, which has no visible representation on the HP49G. That means that we are "not allowed" to use it. But because we don't accept any limits in the usage of our machines, we are going to work with it. Actually the list says nothing more, than a complete description of what to do, in order to get the algebraic object $\text{DERIV}(\text{DERIV}(X^2,X),X)$. It says:

X	Enter X.
2	Enter 2. Now we have the arguments X and 2.
^	Make X^2.
X	Enter X again. Now we have X^2 and X.
'DERIV'	Enter the algebraic object (!) 'DERIV'. Now we have X^2, X and 'DERIV'.
#2d	Enter the system binary #2d. Now we have X^2, X, 'DERIV' and #2d.
•	Apply the function DERIV, to the number of arguments, given by the system binary #2d. Now we have $\text{DERIV}(X^2,X)$.

X	Enter X. Now we have DERIV(X^2,X) and X.
'DERIV'	Enter the algebraic object (!) 'DERIV'. Now we have DERIV(X^2,X), X and 'DERIV'.
#2d	Enter the system binary #2d. Now we have DERIV(X^2,X), X, 'DERIV' and #2d.
•	Apply the function DERIV, to the number of arguments, given by the system binary #2d. Now we have DERIV(DERIV(X^2,X),X).

Since the list contains a complete RPL decomposition of the algebraic object DERIV(DERIV(X^2,X),X), we can use it to detect where the object that we encounter is the algebraic object 'DERIV'. But we need to compare the object in the list with 'DERIV', and we can't enter 'DERIV' ourselves. That is why we enter the dummy DERIV(A,B), decompose it, and put 'DERIV' in the local variable deralg. At this point we also put the invisible FCNAPPLY in the local variable apl, because we are going to need it later in the program. The rest is easy. Whenever we encounter an object different that 'DERIV' or •, we simply evaluate it. This successively builds up our algebraic object. When we encounter a 'DERIV', we don't do anything. When we encounter a •, we turn the object on stack level 3 to an algebraic, because • must have an algebraic object at that level. Then we evaluate •, which builds up expressions of the form someFunction(arg1,arg2 ,argn). Then we check if this expression is actually DERIV(arg1,arg2). If it is, we give arg1 to the new program dCOLLECT, which collects differentials. We keep on doing this until nothing more changes.

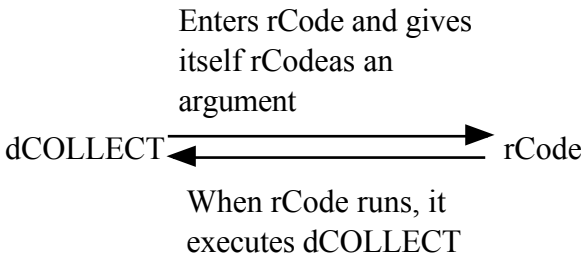
We copy the old dCOLLECT.1 in dCOLLECT and we store the above code in… dCOLLECT itself! This way the program dCOLLECT calls itself over and over again, until it can do nothing more. At the start of the program dCOLLECT we write (additional code is bold):

```
<<
  PUSH                        @Save user's settings
  dn → dv                     @Convert to ∂ notation
  → TERMS                     @return list of terms
  <<                          @All the above code
    …
  >>
  → diffTerms rCode           @Store in locals
  >>
```

We also write in dCOLLECT before the application of the rule of differentiation of an expression multiplied with a constant:

```
    …
END
IF
THEN
   rCode EVAL 1.
ELSE
   0.
END
"C*∂(A) → ∂(C*A)"
…
```

Now we have a program, that gives itself a program as an argument, that in turn calls the program itself again. The process of calling each other ends, when dCOLLECT can't collect anything more. Isn't that amazing?



This is the program dCOLLECT which comes with this document.

Let's try the examples which we examined with dCOLLECT.1.

The expression:

$$F(X)\ \frac{\partial}{\partial X}(F(X))$$

will be converted to

$$\frac{\partial}{\partial X}\left(\frac{1}{2}\ F(X)^2\right)$$

in 17 seconds.

The expression:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)+G(X)\ \frac{\partial}{\partial X}(F(X))+F(X)\ \frac{\partial}{\partial X}(G(X))$$

gets converted to:

$$\frac{\partial}{\partial X}\left(G(X)\ F(X)+\frac{\partial}{\partial X}(F(X))\right)$$

in 97 seconds.

The expression:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)+\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(G(X))\right)+\frac{\partial}{\partial X}(G(X))$$

gets converted to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X)+G(X))+G(X)\right)$$

in 51 seconds.

The expression:

$$\left(3\ F(X)^2+G(X)^2\right)\ \frac{\partial}{\partial X}(F(X))+2\ G(X)\ F(X)\ \frac{\partial}{\partial X}(G(X))$$

goes to:

$$\frac{\partial}{\partial X}\left(G(X)^2\ F(X)+F(X)^3\right)$$

in 116 seconds.

The expression:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))+G(X)\frac{\partial}{\partial X}(F(X))+F(X)\frac{\partial}{\partial X}(G(X))\right)$$

goes to:

$$\frac{\partial}{\partial X}\left(F(X)\ G(X)+\frac{\partial}{\partial X}(F(X))\right)$$

in 97 seconds.

The expression:

$$F(X)^2\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))+2\ F(X)\ \frac{\partial}{\partial X}(F(X))^2\right.$$

goes to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{2}\right)\right)\ F(X)^3$$

in 119 seconds.

The expression:

$$2\ F(X)^2\ \frac{\partial}{\partial X}(F(X))\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)+2\ F(X)^2\ \frac{\partial}{\partial X}(F(X))^3$$

goes to:

$$\frac{\partial}{\partial X}\left(\frac{1}{3}\right)\ \frac{\partial}{\partial X}\left(F(X)^3\right)\ \frac{\partial}{\partial X}(F(X))$$

in 162 seconds.

The expression:

$$F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)+\frac{\partial}{\partial X}(F(X))^2+\frac{\partial}{\partial X}(F(X))$$

goes to

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{2}\right)\right)\ F(X)^2+F(X)$$

in 124 seconds.

The expression:

$$(F(X)+1)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)+\frac{\partial}{\partial X}(F(X))^2$$

goes to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{2}\right)\right)\ F(X)^2+\frac{\partial}{\partial X}(F(X))$$

in 123 seconds.

The expression:

$$F(X)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right)+\frac{\partial}{\partial X}(F(X))^2$$

goes to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{2}\right)\right)\ F(X)^2$$

in 109 seconds.

The expression:

$$\frac{2\ G(X)\ F(X)\ \frac{\partial}{\partial X}(F(X))-F(X)^2\ \frac{\partial}{\partial X}(G(X))}{G(X)^2}$$

goes to:

$$\frac{\partial}{\partial X}\left(\frac{F(X)^2}{G(X)}\right)$$

in 102 seconds.

The expression:

$$\frac{G(X)\ \frac{\partial}{\partial X}(F(X)) - \left(F(X) - G(X)^2\right)\ \frac{\partial}{\partial X}(G(X))}{G(X)^2}$$

goes to

$$\frac{\partial}{\partial X}\left(\frac{F(X)}{G(X)} + G(X)\right)$$

in 110 seconds.

The expression:

$$-\left(3\ F(X)^2 - 3\ F(X)\right)\ \frac{\partial}{\partial X}(F(X))^2$$

goes to:

$$\frac{\frac{\partial}{\partial X}\left(3\ F(X)^2 - 2\ F(X)^3\right)\ \frac{\partial}{\partial X}(F(X))}{2}$$

in 61 seconds.

We do some additional examples.

The expression:

$$3\ F(X)^3\ \frac{\partial}{\partial X}(F(X))$$

goes to:

$$\frac{\partial}{\partial X}\left(\frac{3}{4}\ F(X)^4\right)$$

in 18 seconds.

The expression:

$$3\ F(X)^3\ \frac{\partial}{\partial X}(F(X))^2$$

goes to:

$$\frac{3}{4}\ \frac{\partial}{\partial X}\left(F(X)^4\right)\ \frac{\partial}{\partial X}(F(X))$$

in 23 seconds.

The expression:

$$\left(F(X)^2 + F(X)\right)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}(F(X))\right) + \left(2\ F(X) + 1\right)\ \frac{\partial}{\partial X}(F(X))^2$$

goes to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{2}\ F(X)^2\right) + \frac{\partial}{\partial X}\left(\frac{1}{3}\ F(X)^3\right)\right)$$

in 260 seconds. Here the result could have been further collected to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(\frac{1}{2}\ F(X)^2+\frac{1}{3}\ F(X)^3\right)\right)$$

We let rCode run only after the rule of differentiation of products has been successfully applied. But if you want, you can do the same after the rule of differentiation of sums is applied successfully. This way, when some expression like:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right)+\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(G(X)\right)\right)$$

is successfully collected to:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)+\frac{\partial}{\partial X}\left(G(X)\right)\right),$$

then the sub expression inside the parentheses,:

$$\frac{\partial}{\partial X}\left(F(X)\right)+\frac{\partial}{\partial X}\left(G(X)\right)$$

will itself be collected to:

$$\frac{\partial}{\partial X}\left(F(X)+G(X)\right)$$

This will of course cost even more time.

The expression:

$$F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)\right)$$

goes to:

$$\frac{F(X)}{2}\ \frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\left(F(X)\right)^2\right)$$

in 31 seconds. Try to find out why no complete collection of differential forms was done here.

The expression:

$$2\ U(X)\ G(X)\ F(X)\ \frac{\partial}{\partial X}\left(F(X)\right)+U(X)\ F(X)^2\ \frac{\partial}{\partial X}\left(G(X)\right)+G(X)\ F(X)^2\ \frac{\partial}{\partial X}\left(U(X)\right)$$

goes to:

$$\frac{\partial}{\partial X}\left(U(X)\ G(X)\ F(X)^2\right)$$

in 179 seconds.

Note that the program doesn't contain any explicit implementation of the rule of differentiation of ratios. This is achieved implicitly as a combination of the rule for powers and for products.

There are also other methods to make such a program. One of them will be demonstrated in some future marathon and bases on isomorphism.

We already talked about the fact that the HP49G doesn't provide any commands for conversion of:

$$\frac{\partial\sum_{n=n_0}^{N}F(X,n)}{\partial X}$$

to:

$$\sum_{n=n_0}^{N} \frac{\partial F(X,n)}{\partial X}$$

But it is not hard to program that. We can use pattern matching with the condition that the derivation variable is different from the summation index. We first make a program that smuggles the usage of the command SAME in an algebraic object:

```
<<
  SAME
>>
```

We store it in ALGSAME

Then we program:

```
<<
  {'∂&V(Σ(&n=&n0,&N,&F))'
    'Σ(&n=n0,&N,∂&V(&F))'
    'NOT ALGSAME(&V,&n)'}
  ↑MATCH DROP
>>
```

and we store that in dΣ→Σd.

Enter:

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N} n\ X^n\right)$$

and press dΣ→Σd to get:

$$\sum_{n=1}^{N} \frac{\partial}{\partial X}\left(n\ X^n\right)$$

The derivative:

$$\frac{\partial}{\partial X}\left(n\ X^n\right)$$

inside the sum isn't expanded. If you press EXPAND now, the HP49G will not only expand the derivative to $n\ n\ X^{n-1}$, but will also expand the sum to:

$$\frac{\left(\left(X^2-2\ X+1\right)\ N^2-\left(2\ X-2\right)\ N+X+1\right)\ |X|^N-\left(X+1\right)}{X^3-3\ X^2+3\ X-1}$$

If you don't want that, then you must use another strategy, since smuggling even more RPL in algebraic objects will not work. The customs officers started suspecting us ;-). If you would try to smuggle EXPAND into an algebraic by storing

```
<<
  EXPAND
>>
```

in ALGEXPAND and changing dΣ→Σd to:

```
<<
  {'∂&V(Σ(&n=&n0,&N,&F))'
    'Σ(&n=n0,&N,ALGEXPAND(∂&V(&F)))'
    'NOT ALGSAME(&V,&n)'}
  ↑MATCH DROP
>>
```

then it wouldn't work. If you would enter:

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N} n\ X^n\right),$$

and press dΣ→Σd, then you would get:

$$\sum_{n=1}^{N} \text{ALGEXPAND}\left(\frac{\partial}{\partial X}\left(n\ X^n\right)\right)$$

But it doesn't have to be always pattern matching. Consider for example the program:

```
<<
  → dsum
  <<
    dsum OBJ→ NIP ROT →LST
    IF
      DUP HEAD 4 PICK
      SAME NOT
    THEN
      DUP 4 GET
      4 ROLL 4 ROLL EVAL
      4 SWAP PUT →ALG
    ELSE
      3 DROPN
      dsum
    END
  >>
>>
```

Store that in $d\Sigma \rightarrow \Sigma d2$. Enter again:

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N} n\ X^n\right)$$

and press $d\Sigma \rightarrow \Sigma d2$, to get:

$\sum_{n=1}^{N} n\ n\ X^{n-1}$. Of course this program works only for expressions of the exact form:

$$\frac{\partial}{\partial X}\left(\sum_{n=n_0}^{N} F(X,n)\right)$$

It will not work, or even crash, for any other expression, even if it is just a little bit different, like for example:

$$A\ \frac{\partial}{\partial X}\left(\sum_{n=1}^{N} F(X,n)\right)$$

The other program, $d\Sigma \rightarrow \Sigma d$, is a bit better, since it will also not work for any expression the doesn't contain the pattern:

$$\frac{\partial}{\partial X}\left(\sum_{n=n_0}^{N} F(X,n)\right)$$

but at least it will not crash. Making a program that works in more cases than for some simple patterns, is a bit more difficult, but not very difficult either. We know that the notation $\frac{\partial}{\partial}$ is problematic with Σ but DERIV seems to work better. We must first convert all $\frac{\partial}{\partial}$ in some given expression to DERIV. Then we can expand. The program:

```
<<
  {'∂&V(&F))' 'DERIV(&F,&V)'}
  ↑MATCH DROP EXPAND
>>
```

will do that. Note also that if the HP49G can't handle the sum, then it will return a result that can be questionable. Enter:

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N} F(X,n)\right)$$

and expand. You get $d1\Sigma(n,1,N,F(X,n))$ because the sum:

$$\sum_{n=1}^{N} F(X,n)$$

can't be calculated. But in $d1\Sigma(n,1,N,F(X,n))$ the differentiation is meant for the first rational variable in the parentheses, which is n. We definitely didn't enter:

$$\frac{\partial}{\partial n}\left(\sum_{n=1}^{N} F(X,n)\right)$$

We had a derivative for X. And even worse: In $d1\Sigma(n,1,N,F(X,n))$ the information about the variable of differentiation is completely lost without a trace! Using DERIV instead of ∂ doesn't help either. We seem to have hard problem here. On the one hand we can't be sure that all differentiations of summations will follow the simple pattern:

$$\frac{\partial}{\partial X}\left(\sum_{n=n_0}^{N} F(X,n)\right)$$

and so pattern matching will not always work. On the other hand if we expand trying to convert all differentiations of summations to patterns like $d1\Sigma(n,1,N,F(X,n))$, we lose the variable of differentiation. Is there any way to solve this problem?

If we want to make dCOLLECT good enough for being able to handle arbitrary expressions that contain also sums, then we have add special code that somehow takes care of them. One possibility would be to convert all sums to temporary functions, say $tempFuncA(derVar)$, $tempFuncB(derVar)$, and so on, that contain the variable of differentiation. Then, using EXPAND, we can can convert the given expression to an expression that only contains simple patterns like $d1tempFuncA(X)$. In these patterns the information about the differentiation variable is fully preserved. We can let the code of dCOLLECT run and so collect differential forms with the temporary functions $d1tempFuncA(X)$ instead of the original sums. This will work, because dCOLLECT is already able to handle such abstract expressions, like $d1F(X)$, $d1G(X)$, and thus also $d1tempFuncA(X)$. When we are done with this, we reconvert all $tempFuncA(X)$ back to:

$$\sum_{n=n_0}^{N} F(X,n)$$

and $d1tempFuncA(X)$ back to:

$$\frac{\partial}{\partial X}\left(\sum_{n=n_0}^{N} F(X,n)\right)$$

And so we see that we must keep track of the corresponding expressions $tempFuncA(X)$ and $\sum_{n=n_0}^{N} F(X,n)$, in order to be able to make the backwards conversion afterwards. Let's try first to make a program for conversion of:

$$\frac{\partial}{\partial X}\left(\sum_{n=n_0}^{N} F(X,n)\right)$$

to:

$$\sum_{n=n_0}^{N} \frac{\partial}{\partial X}(F(X,n))$$

Consider the code:

```
<<
  "∂ → DERIV" 1. DISP          @Message
  DO
    {'∂&V(&F)' DERIV(&F,&V)'} @Convert all ∂ to DERIV
    ↑MATCH DROP
  UNTIL
    NOT
  END
  →LST                         @Alg. to list
  'DERIV(A,B)' →LST DUP 3 GET@Create 'DERIV', and the
  SWAP 5 GET                   @invisible APPLY
  {} {} 'X' 64.
  → deralg apl sums            @Store in locals
    tempFuncs dervar fnum
  <<
    "Object" 1. DISP           @Message
    1.
    <<                         @DOSUBS for all objects
      NSUB R→I " of " +        @in list of alg.
      ENDSUB R→I + 2. DISP     @Message
      CASE                     @In case
          DUP deralg SAME      @we have 'DERIV'
        THEN                   @do nothing
        END
          DUP apl SAME         @we have APPLY
        THEN
            ROT 1 →ALG         @Convert object in
            UNROT              @stack level 3 to alg.
            IF                 @If
              PICK3 deralg     @stack level 3 is DERIV
              1. →ALG SAME
            THEN               @Store derivation
              4 PICK           @variable
              'dervar' STO
              EVAL             @Evaluate invis. APPLY
              IF               @If
                sums {} ≠      @sums already found
              THEN
                "tempFunction + (var)"
                2. DISP        @Message
                tempFuncs 1.
                <<             @DOSUBS for all sums
                  IF           @If tempFunction
                    DUP        @doesn't contain
                    dervar     @sub string "dervar,"
                    "," +
                    POS NOT
                  THEN         @then add that sub
                    dervar     @string
                    + ","
                    +
                  END
                >>
                DOSUBS
                'tempFuncs'    @Store in 'tempFuncs'
                STO
              END
            ELSE               @else (stack 3 ≠ DERIV)
              EVAL             @evaluate object
            END
      END
        DUP {Σ} HEAD  SAME     @In case object is Σ
      THEN
        "Σ → tempFunction"
        2. DISP                @Message
        5. →ALG                @Build-up sum
        DUP 'sums' STO+        @Add it to sums
        "tempFunction" fnum    @Make string
        INCR CHR + "(" +       @tempFunctionX where X
        'tempFuncs' STO+       @stands for capital letter
                               @and add to tempFuncs
      END
        EVAL                   @In case nothing of the
                               @above, then EVAL
  END
>>
DOSUBS HEAD                    @*****
IF                             @If
  sums {} ≠                    @we found sums
THEN
```

```
      "tempFunction → alg"
      2. DISP                  @Message
      sums tempFuncs {}
      'tempFuncs' STO          @Store {} in sums
      2.
      <<                       @For all sums, tempFuncs
        1. OVER SIZE 1. - SUB
        ")'" +
        IFERR                  @Try to convert
          OBJ→                 @tempFunction to alg.
        THEN                   @In  case  of  error  get
rid
          DROP2                @of sum and tempFunction
        ELSE
          2. →LIST 1. →LIST    @else keep
          'tempFuncs' STO+     @{sum tempFunction}
        END
      >>
      DOLIST
      IF                       @If we have
        tempFuncs {} ≠         @tempFunctions
      THEN
        "MATCH Σ tempFunction"
        2. DISP                @Message
        tempFuncs 1.           @Match each sum to
        <<                     @its corresponding
          ↑MATCH DROP          @temporary function
        >>
        DOSUBS
      END
    END
    "" 2. DISP                 @Clear display line 2
    dn→dv                      @Run dn→dv
    IF                         @If we have
      tempFuncs {} ≠           @temporary functions
    THEN
      "MATCH tempFunction Σ"
      2. DISP                  @Message
      tempFuncs
      1.                       @Match temporary func.
      <<                       @with sum
        REVLIST ↑MATCH
        DROP
      >>
      DOSUBS
    END
    IF                         @If we have
      tempFuncs {} ≠           @temporary functions
    THEN
      "∂Σ → Σ∂" 2. DISP        @Message
      { '∂&V(Σ(&n=&n0,&N,&F))'
        'Σ(&n=&n0,&N,∂&V(&F))'
        'NOT ALGSAME(&V,&n)' }
      ↑MATCH DROP              @Match ∂Σ to Σ∂
    END
  >>
>>
```

We already had $d\Sigma \rightarrow \Sigma d$ and $d\Sigma \rightarrow \Sigma d2$, so let the program be named $der\Sigma \rightarrow \Sigma der$. (We are running out of names!)

Enter the expression:

$$\frac{\partial}{\partial X}\left(F(X,n)\sum_{n=1}^{N}F(X,n)\right)$$

which doesn't contain the pattern:

$$\frac{\partial}{\partial X}\left(\sum_{n=n_0}^{N}F(X,n)\right)$$

but still can be expanded to:

$$F(X,n)\frac{\partial}{\partial X}\left(\sum_{n=1}^{N}F(X,n)\right)+\sum_{n=1}^{N}F(X,n)\frac{\partial}{\partial X}\left(F(X,n)\right)$$

which does contain the pattern:

$$\frac{\partial}{\partial X}\left(\sum_{n=n_0}^{N} F(X,n)\right)$$

Press der Σ → Σder. The HP49G flashes happily messages, converts, reconverts, and after a while it returns:

$$F(X,n)\ \sum_{n=1}^{N}\frac{\partial}{\partial X}\left(F(X,n)\right)+\sum_{n=1}^{N}F(X,n)\ \frac{\partial}{\partial X}\left(F(X,n)\right)$$

Notice that the derivation is now *inside* the summation. The program worked. The other two programs, dΣ → Σd and dΣ → Σd2, would live the expression:

$$\frac{\partial}{\partial X}\left(F(X,n)\ \sum_{n=1}^{N}F(X,n)\right)$$

unchanged. Notice also that the code of the program up to the point marked with ***** in the program listing, strongly resembles the part rCode of dCOLLECT.

With almost the same code of derΣ → Σder we can also enhance our dCOLLECT, in order to be able to collect differential forms. At the point in the program, where we call dn → dv, we could also call the current version of dCOLLECT. That means that we can make yet another new version that handles also sums. But we can also combine the two functionalities of collection of differential forms and of converting:

$$\frac{\partial}{\partial X}\left(\sum_{n=n_0}^{N} F(X,n)\right)$$

to:

$$\sum_{n=n_0}^{N}\frac{\partial}{\partial X}\left(F(X,n)\right)$$

in a single program. This program takes an expression with differential forms from stack level 2, *and* a 1 or 0 from stack level 1. If the argument on stack level 1 is a 1, the program collects differential forms. If it is a 0, it just converts:

$$\frac{\partial}{\partial X}\left(\sum_{n=n_0}^{N} F(X,n)\right)$$

to:

$$\sum_{n=n_0}^{N}\frac{\partial}{\partial X}\left(F(X,n)\right)$$

by expanding the differential forms. We just have to make minor modifications in derΣ → Σder. At the start of the program we write (bold type face):

```
<<
  SWAP
  "∂ → DERIV" 1. DISP          @Message
  DO
    {'∂&V(&F)' DERIV(&F,&V)'} @Convert all ∂ to DERIV
    ↑MATCH DROP
  UNTIL
    NOT
  END
  →LST                         @Alg. to list
  SWAP
  'DERIV(A,B)' →LST DUP 3 GET@Create 'DERIV', and the
  SWAP 5 GET                   @invisible APPLY
  {} {} 'X' 64.
  → dColFlag  deralg apl sums     @Store in locals
    tempFuncs dervar fnum
```

At the middle of the program we add:

```
  ..........
"" 2. DISP                  @Clear display line 2
IF                          @If
  dColFlag                  @the user entered 1
THEN                        @then do collection
  dCOLLECT                  @of diff. forms.
ELSE                        @else
  dn → dv                   @Run dn → dv
END
IF                          @If we have
  tempFuncs {} ≠            @temporary functions
THEN
  "MATCH tempFunction Σ"
  ...........
```

This is the program dCOLEX that comes with this document. Let's try it. Enter:

$$\frac{\partial}{\partial X}\left(F(X,n)\ G(X)\right)$$

then 0 (for expansion of differential forms) and press dCOLEX. The result is the expression:

$$G(X)\ \frac{\partial}{\partial X}\left(F(X,n)\right)+F(X,n)\ \frac{\partial}{\partial X}\left(G(X)\right)$$

If you now enter 1 (for collection of differential forms) and press dCOLEX again, then you get:

$$\frac{\partial}{\partial X}\left(G(X)\ F(X,n)\right),$$

the expression we started with. But the improvement becomes visible when working with sums. Enter:

$$\frac{\partial}{\partial X}\left(G(X)\ \sum_{n=1}^{N}F(X,n)\right)$$

then 0, and press dCOLEX. You get the expanded result:

$$G(X)\ \sum_{n=1}^{N}\frac{\partial}{\partial X}\left(F(X,n)\right)+\frac{\partial}{\partial X}\left(G(X)\right)\ \sum_{n=1}^{N}F(X,n)$$

where the derivation is brought *inside* the parentheses of the sum. Now, it would be good if the collection of differential forms would already work, but we still need a tiny modification. Before calling dCOLLECT we must convert all patterns:

$$\sum_{n=1}^{N}\frac{\partial}{\partial X}\left(F(X,n)\right)$$

to:

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N}F(X,n)\right)$$

We will modify slightly the code in dCOLEX almost since everything we need is already there. At the beginning of the program, after the first SWAP, we add:

```
"Σ ∂ → ∂ Σ" 1. DISP
DO
  {'Σ(&n=&n0,&N,∂&V(&F))' '∂&V(Σ(&n=&n0,&N,&F))'
    'NOT ALGSAME(&V,&n) } ↑MATCH
UNTIL NOT
END
```

Now, having:

$$G(X)\ \sum_{n=1}^{N}\frac{\partial}{\partial X}(F(X,n))+\frac{\partial}{\partial X}(G(X))\ \sum_{n=1}^{N}F(X,n)$$

or:

$$G(X)\ \frac{\partial}{\partial X}\left(\sum_{n=1}^{N}F(X,n)\right)+(G(X))\ \sum_{n=1}^{N}F(X,n)$$

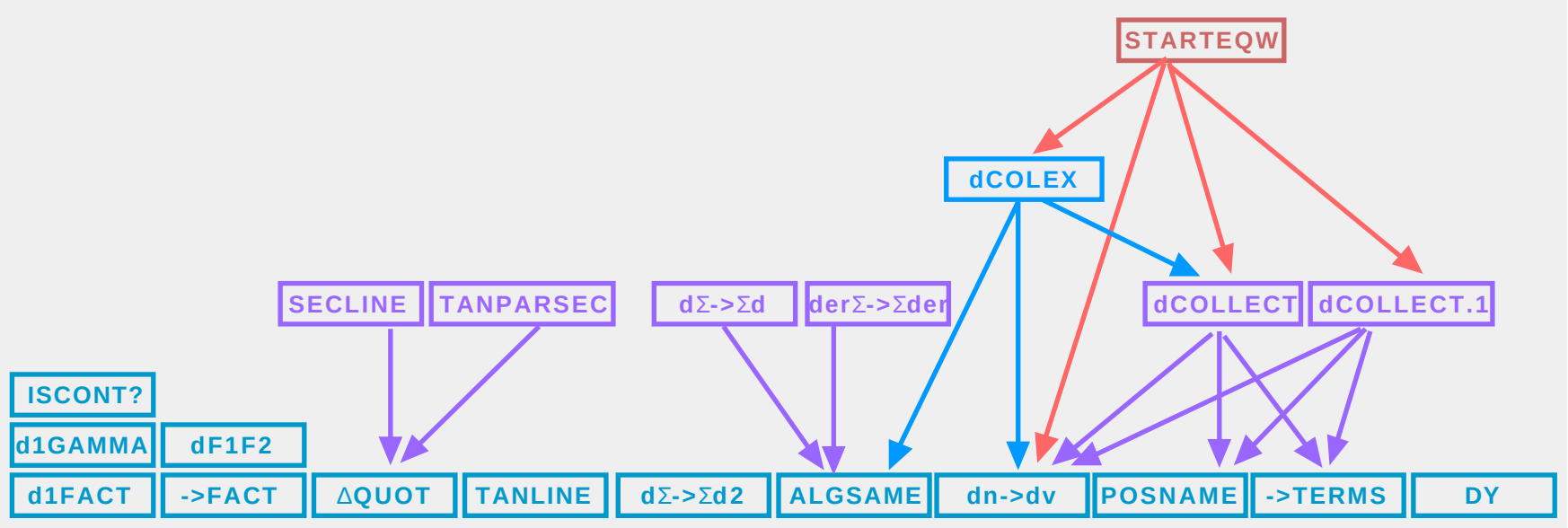
on stack level 1, you can enter a 1, and press dCOLEX to get:

$$\frac{\partial}{\partial X}\left(G(X)\ \sum_{n=1}^{N}F(X,n)\right)$$

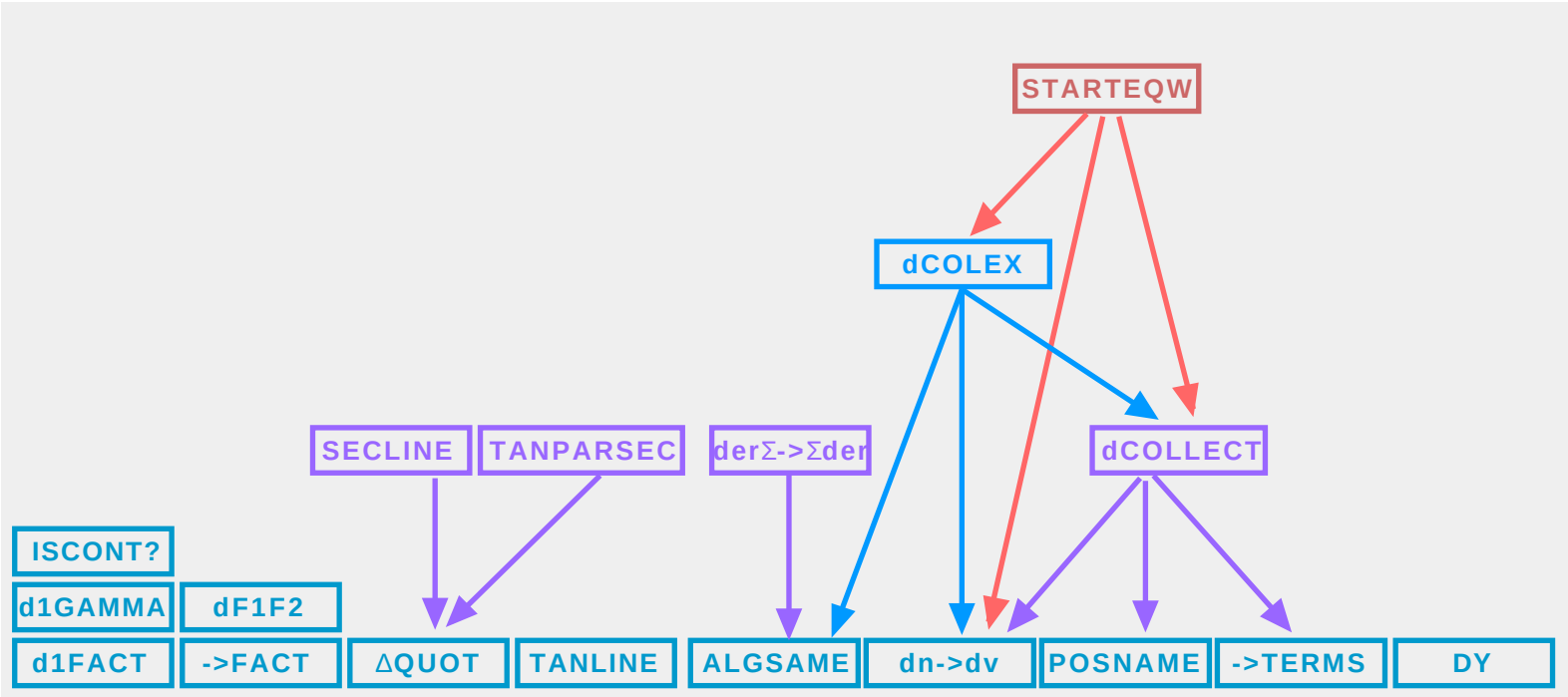
The program still works with the examples that we had on the previous page. If it doesn't, then call Trabakoulas and tell him the story. He will be glad to "suggest" me to do more programming exercises. ;-)

I think that it is time now to take a look at the interdependencies of our programs, because we had so many of them, and we are going to lose track, who is using whom, and what we need and we can through away.

We don't need necessarily dΣ→Σd and dΣ→Σd2. These two programs can be purged if you don't need them. I only include them with this document for studying purposes. (And to spare you the typing on the hard keys of the HP49G ;-)) Also, dCOLLECT.1 is not needed. The programs dCOLEX and dCOLLECT do that work better. You could keep dCOLLECT, though dCOLEX does the same and more, because dCOLLECT is faster, since it doesn't include special code for sums. So perhaps you can use it for faster results, when no sums are involved. If you purge dΣ→Σd, dΣ→Σd2, and dCOLLECT.1, then the program structure becomes a bit simpler. (The picture of the simpler program structure is on the next page.) The two programs ISCONT? and →TERMS come from the Sequences, Series and Limits Marathon. Their complete documentation is there, but I include them in the files of this marathon for convenience.



If we bring the main ingredients of the programs dn → dv, dCOLLECT derΣ → Σder, or dCOLEX, then we see that they use list processing very extensively. Algebraics objects are transformed to the corresponding RPL lists, and then each of the list objects is examined and used individually. The programs work in many cases. Of course they will not work in other cases. But the main picture that we get from them is that we are actually process lists, in this case the particular RPL lists that are equivalent to the algebraic objects of the HP49G. Without doubts the lists are a very powerful object n the HP49G, because the are somewhere between algebraic objects, programs, and data. Having commands like → LST, → ALG and similar, we are able to start with an algebraic object, convert it to a list, jumping thus in the world of listoids with all their special capabilities, transform the list to a new list which corresponds to some other algebraic objects, and finally jump back to the world of algebraicoids with their special capabilities. When we transform the list, using our rules of programming, we actually are performing operations on algebraic objects in their list form. You can imagine some virtual algebraic object, which experiences the corresponding changes, when we somehow operate on its list form. For example, take the algebraic object $X+Y$, and its RPL list form $\{X\ \ Y\ \ +\}$. If we exchange the elements X and Y in the list, we are actually transforming $X+Y$ to $Y+X$. There are of course countless ways to exchange elements X and Y of the list, but all these possible algorithms, programs, name them what you like, are actually doing the same. They use the commutative property of addition. Any hypothetical built-in command COMMUTEPLUS, would correspond to our element exchange algorithm. But, not all elements exchanges would correspond to commutation of the operands of addition. In the list we can also exchange the elements Y and $+$, transforming it to $\{X\ \ +\ \ Y\}$, which is no more the mirror picture of some valid algebraic object in the world of the listoids. The list $\{X\ \ +\ \ Y\}$, would be like trying to enter '$Y(X,+)$' which isn't possible on the HP49G. (Well, usually at least. ;-)) This shows us that we have ways



on the HP49G, to extend the possible algebraic objects, to all those which are representable through valid lists, but not through valid algebraic expressions. ("Valid" means valid for the HP49G.) For example, the object 'Y(X,+)', represented by {X + Y}, could mean some function Y, that acts upon two arguments, one of which is itself a function! We can use it for any program, that we code accordingly to what the function Y does with its arguments. We can only not represent it using algebraic syntax, but that's the only limitation. So we see that the world of listoids seems to be somehow more extended, bigger than the world of algebraicoids. Imagine the possibilities. (We already have experienced some of these possibilities, think again of our programs.)

But one of the main disadvantages of the lists are, that they are harder to understand by simply looking at them. Enter some lengthy expression using derivatives, sums and whatever you want, and take a look at it. A simple short look in the EQW. If you think that it is not very readable, press →LST, and think about the readability of the RPL list. Perhaps because of years in school, perhaps because of built-in preference of humans for algebraic syntax, we seem to be able to understand algebraics (by looking at them) better than RPL. (Or did anybody read any book of physics, that says that the one dimensional Schrödinger equation is ħ 2 m / NEG Ψ x ∂ x ∂ E Ψ = instead of $-\frac{\hbar}{2\,m}\,\frac{\partial^2\Psi}{\partial x^2} = E\,\Psi$?) Don't think that I am a fan of algebraic syntax on calculators, or on any other CAS. I am a fan of the *algebraic syntax as it is used in mathematics*, and that's (often) quite different from that used on computer algebra systems. Compare the mathematics formula $-\frac{\hbar}{2\,m}\,\frac{\partial^2\Psi}{\partial x^2} = E\,\Psi$ with the calculator formula −(ħ /(2 m) ∂x(∂x(Ψ))) = E Ψ to understand what I mean. They are way not the same. Only when some input software like the EQW is available, somebody can say that algebraic is like it was meant "on paper". Otherwise the similarity exists only on the prospects of the device/CAS.

Why do I say the above things? Well, the lists may be as good and powerful as they want, but they don't have the transparency of using an algebraic, at least on the HP49G. Algebraic objects and the commands available for them, are easier to understand. Somehow clearer. So, isn't there any way to do, for example, collection of differential forms, avoiding lists and their difficulties when it comes to understanding? Can't we do the same, like what dΣ → Σd or dCOLLECT does, but using only algebraic objects and the commands for them? Well, there is! But in order to understand this (marvellous!) way, we have to take a closer look to our old good friends ↑ MATCH and ↓ MATCH again. We have met them in the Trigonometry Marathon for the first time. And their power is waaaaay from being completely known. What follows is a first short journey into the depths of pattern matching on the HP49G. I think that many many journeys will follow in the next marathons.

Let's bring in mind again, why we avoided using pattern matching for collecting differential forms in expressions that contain sums. The problem was, that we can't be sure that all sums will be in a few simple patterns, which we can then convert to other patterns with pattern matching. In order to convert the whole expression in another expression, in which only some few simple patterns with sums appear, we have to expand it first. But then all expressions of the form:

$$\frac{\partial}{\partial X}\left(\sum_{n=n_0}^{N} F(X,n)\right)$$

will be converted to d1Σ(n,n0,N,F(X,n)) and we lose the information about the variable of derivation. That was the reason that we decided to first convert every sum in a temporary function tempFunctionA(X), tempFunctionB(X) and so on. This way any expression like:

$$\frac{\partial}{\partial X}\left(\sum_{n=n_0}^{N} F(X,n)\right)$$

gets converted to:

$$\frac{\partial}{\partial X}\left(\text{tempFunctionA}(X)\right)$$

and then we are able to expand, because then the result is $\text{d1tempFunctionA}(X)$. After this we apply our algorithms for collection of differential forms, and we reconvert all temporary functions back to sums. One small but important detail of the procedure is that each distinct sum should be converted to a distinct temporary function, in order to keep the information that different sums are involved. This is why we can't use pattern matching. If we have an expression, like for example:

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N} F(n,X) + \sum_{m=1}^{N} G(m,X)\right)$$

and we match all patterns $\sum_{\&n=\&n0}^{\&N} \&F$ to $\text{tempFunctionA}(X)$, then the expression:

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N} F(n,X) + \sum_{m=1}^{N} G(m,X)\right)$$

will be converted to:

$$\frac{\partial}{\partial X}\left(\text{tempFunctionA}(X) + \text{tempFunctionA}(X)\right)$$

which is wrong since there were two different sums in the original expression, but only a single temporary function in the result.

But if we were able to expand the differential forms in the expression completely but without using EXPAND, then we would at least be sure, that the only differential form where sums appear, would be of the form:

$$\frac{\partial}{\partial X}\left(\sum_{n=n_0}^{N} F(X,n)\right)$$

If we know that this is the only possible pattern, we are half the way through. Of course, we can't expect that a single pattern matching could do that. But using many of them, in a reasonable order, it is possible to do what we want. Consider the program:

```
<<
  DO
    DUP  → TERMS
    0 + Σ LIST
    "∂(A/B) → (B*∂A-A*∂B)/B^2"
    1. DISP
    DO                        @Ratio rule
      {'∂&V(&A/&B)' '(&B*∂&V(&A)-&A*∂&V(&B))/&B^2' }
      ↑ MATCH
    UNTIL
      NOT
    END
     → TERMS 0 + Σ LIST
    "∂(A+B) → ∂A+∂B"
    1. DISP
    DO                        @Sums rule
      {'∂&V(&A+&B)' '∂&V(&A)+∂&V(&B)' }
      ↑ MATCH
    UNTIL
      NOT
    END
    "∂(Const*A) → Const*∂A"
    1. DISP
    DO                        @Constant rule
      {'∂&V(&C*&A)' '&C*∂&V(&A)' 'NOT POSNAME(&C,&V)' }
      ↑ MATCH
    UNTIL
      NOT
```

```
    END
    "∂(A*B)→B*∂A+A*∂B"
    1. DISP
    DO                      @Product rule
      {'∂&V(&A*&B)' '&B*∂&V(&A)+&A*∂&V(&B)' }
      ↑MATCH
    UNTIL
      NOT
    END
    "∂(A^n)→n*A^(n-1)*∂A"
    1. DISP
    DO                      @Power rule
      {'∂&V(&A^&n)' '&n*&A^(&n-1)*∂&V(&A)'
      'NOT POSNAME(&n,&V)'}
      ↑MATCH
    UNTIL
      NOT
    END
    "∂Σ→Σ∂"                 @All other expansions of
    1. DISP                 @diff. forms are done. So we
    DO                      @start converting sums
      {'∂&V(Σ(&n=&n0,&N,&F))'
      'Σ(&n=&n0,&N,∂&V(&F))'
      'NOT ALGSAME(&V,&n) }
      ↑MATCH
    UNTIL
      NOT
    END
  UNTIL
    SWAP OVER SAME
  END
>>
```

This is the program PATdΣ → Σd. Let's see it in action. Enter:

$$\frac{\partial}{\partial X}\left(G(X,n)\sum_{n=1}^{N}F(X,n)\right)$$

and press PATdΣ → Σd. The program returns:

$$\sum_{n=1}^{N}F(X,n)\ \frac{\partial}{\partial X}\left(G(X)\right)+G(X)\ \sum_{n=1}^{N}\frac{\partial}{\partial X}\left(F(X,n)\right),$$

the correct result. Notice that we didn't use EXPAND a single time. Notice also that we used → TERMS, which itself doesn't convert the patterns:

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N}F(X,n)\right)$$

to $d1\Sigma(n,n_0,N,F(X,n))$. Try some examples yourself. I'm sure that you will find expressions that the program can't handle, but one thing is sure. This program is a much clearer and easier to understand, than the others. No DOSUBS of DOSUBS and procedures buried under nested levels of loops. No tricks and dummies from which we extract objects that we can't create directly. Just DO-loops, nicely ordered one after the other, each one of which simply repeats a single kind of pattern matching, until nothing more happens. And the whole thing wrapped in an outer DO-loop which runs again and again, until nothing changes.

Now that we have this program, we know that whatever the form is, in which differential forms are present in some expression, when the program ends all derivatives of sums will be in the form of a single simple pattern:

$$\sum_{\&n=\&n0}^{\&N}\frac{\partial}{\partial\,\&X}(\&F)$$

This contains all information that we need in order to convert it to a temporary function. So we can hope that we will be able to use again pattern matching for collecting differential forms including differential forms that contain sums. We can convert all patterns:

$$\sum_{\&n=\&n0}^{\&N} \frac{\partial}{\partial \&X}(\&F)$$

to:

$$\frac{\partial}{\partial \&X}\left(\sum_{\&n=\&n0}^{\&N} \&F\right)$$

then convert all patterns:

$$\frac{\partial}{\partial \&X}\left(\sum_{\&n=\&n0}^{\&N} \&F\right)$$

to patterns:

$$\frac{\partial}{\partial \&X}\left(\text{tempFunction}(\&X)\right)$$

then apply dCOLLECT, and then convert all patterns tempFunction(&X) back to:

$$\sum_{\&n=\&n_0}^{\&N} \&F$$

The principle is clear, but…

As we have seen, if we try to match each pattern:

$$\frac{\partial}{\partial \&X}\left(\sum_{\&n=\&n0}^{\&N} \&F\right)$$

to , say:

$$\frac{\partial}{\partial \&X}\left(\text{REPL}\Sigma\left(\sum_{\&n=\&n0}^{\&N} \&F\right)\right)$$

having programmed REPLΣ before, as a program that takes the sum and creates replacements, like tempFunctionA(X), then we have two small problems. First of all, the replacements have to be different from each other for different sums, and second, if we match some expression like:

$$\frac{\partial}{\partial X}\left(\sum_{n=n0}^{N} F(X)\right)$$

using the pattern list:

$$\left\{\frac{\partial}{\partial \&X}\left(\sum_{\&n=\&n0}^{\&N} \&F\right) \quad \frac{\partial}{\partial \&X}\left(\text{repl}\left(\sum_{\&n=\&n0}^{\&N} \&F, \&n, \&n+1\right)\right)\right\}$$

then we get the result:

$$\frac{\partial}{\partial X}\left(\text{REPL}\Sigma\left(\sum_{n=n0}^{N} F(X)\right)\right)$$

where the replacement program wasn't evaluated. So we have to code the program REPLΣ in a way, that is assigns distinct temporaryFunction names to distinct sums. And we have to expand the whole expression, in order to evaluate all sub expressions:

$$\frac{\partial}{\partial X}\left(\text{REPL}\Sigma\left(\sum_{n=n0}^{N} F(X)\right)\right)$$

Since the inner most nested sub expressions are evaluated first, we can be sure that:

REPLΣ$\left(\sum_{n=n0}^{N} F(X)\right)$

will be evaluated before the differentiation, producing the result:

$$\frac{\partial}{\partial X}\left(\text{tempFunctionA}(X)\right)$$

which will be further evaluated to d1tempFunctionA(X). Let's first make the program REPLΣ. It just has to return an algebraic object, but can do anything else in-between.

```
<<
   →  sum derVar                @Store in locals
  <<
    IF
      VARS SLIST POS NOT   @If SLIST doesn't exist
    THEN                   @then
      {} 'SLIST' STO       @create SLIST and RLIST
      {} 'TLIST' STO
      64. 'nTemp' STO      @initialise nTemp
    END
    IF
      SLIST sum POS NOT    @If sum not in SLIST
    THEN                   @then
      'SLIST' sum STO+     @Add sum to SLIST
      'TLIST'
      "'tempFunction"      @Create distinct temporary
      'nTemp' INCR CHR +   @function for replacement
      "(" + derVar + ")'" +
      + OBJ→  DUP UNROT    @Copy of tempFunction for
                           @replacement in expression
      STO+                 @Add tempFunction to TLIST
    END
  >>
>>
```

This program takes a sum and a derivation variable, and updates the list of sums SLIST and the list of temporary replacement functions TLIST. It returns a distinct temporary replacement function for every sum that it receives as argument.

Now we make PATdCOLLECT, the program for collection of differential forms that uses pattern matching. Actually we can use much of the code of the program PATdΣ → Σd.

```
<<
  DO
    DUP →TERMS
    0 + Σ LIST
    "∂(A/B)→(B*∂A-A*∂B)/B^2"
    1. DISP
    DO                        @Ratio rule
      {'∂&V(&A/&B)' '(&B*∂&V(&A)-&A*∂&V(&B))/&B^2' }
      ↑MATCH
    UNTIL
      NOT
    END
    →TERMS 0 + Σ LIST
    "∂(A+B)→∂A+∂B"
    1. DISP
    DO                        @Sums rule
      {'∂&V(&A+&B)' '∂&V(&A)+∂&V(&B)' }
      ↑MATCH
    UNTIL
      NOT
    END
    "∂(Const*A)→Const*∂A"
    1. DISP
    DO                        @Constant rule
      {'∂&V(&C*&A)' '&C*∂&V(&A)' 'NOT POSNAME(&C,&V)' }
      ↑MATCH
    UNTIL
      NOT
    END
    "∂(A*B)→B*∂A+A*∂B"
    1. DISP
```

```
  DO                      @Product rule
    {'∂&V(&A*&B)' '&B*∂&V(&A)+&A*∂&V(&B)' }
    ↑MATCH
  UNTIL
    NOT
  END
  "∂(A^n)→n*A^(n-1)*∂A"
  1. DISP
  DO                      @Power rule
    {'∂&V(&A^&n)' '&n*&A^(&n-1)*∂&V(&A)'
    'NOT POSNAME(&n,&V)'}
    ↑MATCH
  UNTIL
    NOT
  END
UNTIL
  SWAP OVER SAME
END                       @Except the part ∂Σ→Σ∂
                          @and conversion of derivat.
                          @of sums, this is the code
                          @of PAT∂Σ→Σ∂.
"Σ∂ → ∂Σ"
1. DISP
DO                        @Match Σ∂ to ∂Σ
  {'Σ(&n=&n0,&N,∂&V(&F))'
   '∂&V(Σ(&n=&n0,&N,&F))
   'NOT ALGSAME(&V,&n)'}
  ↑MATCH
UNTIL
  NOT
END

"∂Σ → ∂REPLΣ"
1. DISP
DO                        @Match ∂Σ to ∂(REPLΣ(Σ))
  {'∂&V(Σ(&n=&n0,&N,&F))'
   '∂&V(REPLΣ(Σ(&n=&n0,&N,&F),&V))}
  ↑MATCH
UNTIL
  NOT
END

EXPAND                    @Create replacements

IF
  SLIST {} ≠
THEN
  "Σ → tempFunct"
  1. DISP
  SLIST TLIST 2.          @Match Σ to replacement
  <<                      @function
    2. ->LIST MATCH DROP
  >> DOLIST
END

dCOLLECT                  @Collect diff. forms

IF
  SLIST {} ≠
THEN
  "tempFunc → Σ"
  1. DISP
  TLIST SLIST 2.          @Match replacement function
  <<                      @back to original Σ
    2. ->LIST MATCH DROP
  >> DOLIST

  "d1tempFunc → ∂Σ"
  1. DISP
  TLIST 1.
  <<                      @Create list of d1tempFunct
    ->STR 2. OVER SIZE
    SUB "'d1" SWAP +
    OBJ→
  >> DOSUBS

  SLIST TLIST 2.          @Create list of ∂Σ
  <<
    OBJ-> DROP2
    SWAP ->STR 2. OVER
```

```
      SIZE 1. - SUB
      "'∂" ROT + "(" +
      SWAP + ")'" +
      OBJ->
    >> DOLIST
    2.
    <<                          @Match d1tempFunc to ∂Σ
      2. ->LIST ↑MATCH DROP
    >> DOLIST
    { SLIST TLIST nTemp }
    PURGE
  END
>>
```

Notice here the following remarkable thing. When we match all patterns of the form:

$$\frac{\partial}{\partial X}\left(\sum_{n=n0}^{N} F(X)\right)$$

to:

$$\frac{\partial}{\partial X}\left(\text{REPL}\Sigma\left(\sum_{n=n0}^{N} F(X)\right)\right)$$

we do only a formal replacement. The real replacement follows much later in two stages. The first stage is the evaluation of the expression which creates the list of sums SLIST and the list of temporary replacement functions TLIST . The second stage is the following pattern matching, which matches not

$$\frac{\partial}{\partial X}\left(\sum_{n=n0}^{N} F(X)\right)$$

but rather $\sum_{n=n0}^{N} F(X)$ to temporary replacement functions. This is a very important thing to do! We don't want only

$$\frac{\partial}{\partial X}\left(\sum_{n=n0}^{N} F(X)\right)$$

but also $\sum_{n=n0}^{N} F(X)$ to be matched to temporary replacement functions, in order to retain the original structure of the expression regarding differential forms. If for example we have:

$$\sum_{n=1}^{N} F(X,n)\ \frac{\partial}{\partial X}\left(G(X)\right) + G(X)\ \frac{\partial}{\partial X}\left(\sum_{n=1}^{N} F(X,n)\right)$$

and we replace only the sums in differential forms, then we will get:

$$\sum_{n=1}^{N} F(X,n)\ \frac{\partial}{\partial X}\left(G(X)\right) + G(X)\ \frac{\partial}{\partial X}\left(\text{tempFunctionA}(X)\right)$$

in which dCOLLECT can't collect any differential forms, simply because there is nothing to collect. But if we also replace the sums that are not in differential forms, we get the expression:

$$\text{tempFunctionA}(X)\ \frac{\partial}{\partial X}\left(G(X)\right) + G(X)\ \frac{\partial}{\partial X}\left(\text{tempFunctionA}(X)\right)$$

which dCOLLECT can convert to:

$$\frac{\partial}{\partial X}\left(\text{tempFunctionA}(X)\ G(X)\right)$$

and in which we can match tempFunctionA(X) back to $\sum_{n=1}^{N} F(X,n)$, getting the result:

$$\frac{\partial}{\partial X}\left(\sum_{n=1}^{N} F(X,n)\right)\ G(X)$$

This is one example of *delayed evaluation and delayed pattern matching*, a very powerful possibility that we have for formula manipulation.

Let's try the program PATdCOLLECT. Enter the expression:

$$\sum_{n=1}^{N} F(X,n)\ \frac{\partial}{\partial X}\left(G(X)\right) + G(X)\ \sum_{n=1}^{N} \frac{\partial}{\partial X}\left(F(X,n)\right)$$

and press PATdCOLLECT. The program returns:

$$\frac{\partial}{\partial X}\left(G(X)\ \sum_{n=1}^{N} F(X,n)\right),$$

the collected form.

In the program PATdCOLLECT much of the code is identical to the code of PATdΣ → Σd. Again we can combine the functionalities of the two programs in one program, PATdCOLEX, that needs an additional argument, a 0 or a 1, that specifies if we want conversion of ∂Σ to Σ∂ (i.e. expansion of differential forms), or collection of differential forms. The following listing is PATdCOLEX.

```
<<
  → dColFlag
  <<
    DO
      DUP → TERMS
      0 + Σ LIST
      "∂(A/B) → (B*∂A-A*∂B)/B^2"
      1. DISP
      DO                    @Ratio rule
        {'∂&V(&A/&B)'
        '(&B*∂&V(&A)-&A*∂&V(&B))/&B^2' }
        ↑ MATCH
      UNTIL
        NOT
      END
      → TERMS 0 + Σ LIST
      "∂(A+B) → ∂A+∂B"
      1. DISP
      DO                    @Sums rule
        {'∂&V(&A+&B)' '∂&V(&A)+∂&V(&B)' }
        ↑ MATCH
      UNTIL
        NOT
      END
      "∂(Const*A) → Const*∂A"
      1. DISP
      DO                    @Constant rule
        {'∂&V(&C*&A)' '&C*∂&V(&A)'
        'NOT POSNAME(&C,&V)' }
        ↑ MATCH
      UNTIL
        NOT
      END
      "∂(A*B) → B*∂A+A*∂B"
      1. DISP
      DO                    @Product rule
        {'∂&V(&A*&B)' '&B*∂&V(&A)+&A*∂&V(&B)' }
        ↑ MATCH
      UNTIL
        NOT
      END
      "∂(A^n) → n*A^(n-1)*∂A"
      1. DISP
      DO                    @Power rule
```

```
  {'∂&V(&A^&n)' '&n*&A^(&n-1)*∂&V(&A)'
  'NOT POSNAME(&n,&V)'}
  ↑MATCH
UNTIL
  NOT
END
IF
  dColFlag          @If we want collection.
THEN
  "Σ∂ → ∂Σ"
  1. DISP
  DO                @Match Σ∂ to ∂Σ
    {'Σ(&n=&n0,&N,∂&V(&F))'
     '∂&V(Σ(&n=&n0,&N,&F))
     'NOT ALGSAME(&V,&n)'}
    ↑MATCH
  UNTIL
    NOT
  END
  "∂Σ → ∂REPLΣ"
  1. DISP
  DO                @Match ∂Σ to ∂(REPLΣ(Σ))
    {'∂&V(Σ(&n=&n0,&N,&F))'
    '∂&V(REPLΣ(Σ(&n=&n0,&N,&F),&V))}
    ↑MATCH
  UNTIL
    NOT
  END
  EXPAND              @Create replacements
  IF
    SLIST {} ≠
  THEN
    "Σ → tempFunct"
    1. DISP
    SLIST TLIST 2.    @Match Σ to replacement
    <<                @function
      2. ->LIST MATCH DROP
    >> DOLIST
  END
  dCOLLECT            @Collect diff. forms
  IF
    SLIST {} ≠
  THEN
    "tempFunc → Σ"
    1. DISP
    TLIST SLIST 2.    @Match replacement
    <<                @function back to
      2. ->LIST MATCH @original Σ
      DROP
    >> DOLIST
    "d1tempFunc → ∂Σ"
    1. DISP
    TLIST 1.
    <<                @Create list of
      ->STR 2. OVER   @d1tempFunct
      SIZE SUB "'d1"
      SWAP + OBJ→
    >> DOSUBS
    SLIST TLIST 2.    @Create list of ∂Σ
    <<
      OBJ-> DROP2
      SWAP ->STR 2. OVER
      SIZE 1. - SUB
      "'∂" ROT + "(" +
      SWAP + ")'" +
      OBJ->
    >> DOLIST
    2.
    <<                @Match d1tempFunc to ∂Σ
      2. ->LIST ↑MATCH DROP
    >> DOLIST
    { SLIST TLIST nTemp }
    PURGE
  END
ELSE                  @we want expansion
  "∂Σ→Σ∂"             @All other expansions of
  1. DISP             @diff. forms are done. So
  DO                  @we start converting sums
    {'∂&V(Σ(&n=&n0,&N,&F))'
    'Σ(&n=&n0,&N,∂&V(&F))'
    'NOT ALGSAME(&V,&n) }
```

```
          ↑ MATCH
        UNTIL
          NOT
        END
      END
    UNTIL
      SWAP OVER SAME
    END
  >>
>>
```

A small test to see that it works. Enter:

$$\frac{\partial}{\partial X}\left(G(X,n)\ \sum_{n=1}^{N} F(X,n)\right)$$

then 0, and press PATdCOLEX to get the expanded form:

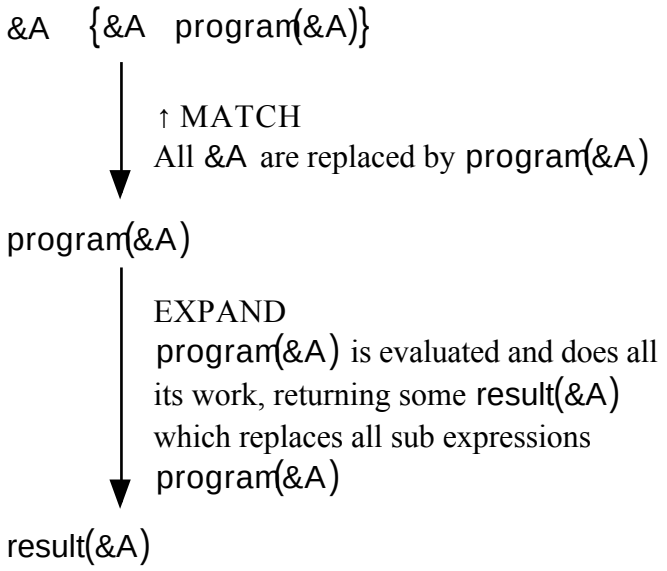
$$G(X)\ \sum_{n=1}^{N}\frac{\partial}{\partial X}\left(F(X,n)\right)+\sum_{n=1}^{N}F(X,n)\ \frac{\partial}{\partial X}\left(G(X)\right)$$

Now enter 1 and press again PATdCOLEX to get back to:

$$\frac{\partial}{\partial X}\left(G(X,n)\ \sum_{n=1}^{N} F(X,n)\right),$$

the form we started with.

The above might be nice, but the pattern matching commands have even more depths, which we can explore. As we already saw, when pattern matching is used, to replace some give pattern with the sub expression program(patternArgs), where program is a program that we wrote previously, and patternArgs are arguments built-up using the patterns, then the returned algebraic object contains sub expressions program(patternArgs), i.e. the program isn't automatically evaluated when the pattern matching is successful. We have to expand explicitly afterwards, to put the result of program(patternArgs) in place of program(patternArgs) in the algebraic object. The program has to be a program that returns one single algebraic object, or anything else that is allowed in algebraics, like for example a number. The whole procedure works like on the picture at the right. It has to be at least two stages, because program is not automatically evaluated at the moment of pattern matching. But as we also saw, if we use pattern matching with a list of three elements, then the third element is used as a condition for pattern matching. The pattern matching commands first check to see if the patterns that must be matched exist. If they don't exist, the algebraic object is returned to the stack unchanged. *But if the patterns do exist, then the condition for pattern matching is evaluated*, and the matching is done if the condition is true. Aha! You see where it goes. Couldn't we make a program, name it test, that not only returns a truth value, but also does other work *before it returns the truth value*? Of course we can! Let's start with easy examples. First of all, there are no special objects for true and false in user RPL. Any number different than 0 is true, and 0 is false. (So the HP49G is an extraordinary truth loving machine, since there are so many more numbers that are equivalent to true. ;-)) We make first the



program TEST1.

```
<<
    IF                      @If
      VARS 'nIter' POS NOT @variable nIter doesn't
                            @exist
    THEN                    @then
      0. 'nIter' STO        @initialise it
    END
    'nIter' INCR            @Increment nIter
                            @and
    1. DISP                 @display current
                            @value
    1.                      @Return        1.
(true)
>>
```

It doesn't use any arguments and always evaluate to true. That means that if the patterns exist, that we specify in the pattern list, the pattern matching will be always performed.

Enter $F(F(F(X)))$ and $\{\&F(\&X)\ \ \&X-1\ \ TEST1\}$.
Now use the command ↑ MATCH to get $F(X-1)-1$.
Did you see what happened in addition? The program TEST1 was evaluated and it counted how many time it was evaluated, that is how many passes ↑ MATCH did. As we already saw, ↑ MATCH does not only one pattern matching but it starts from the inner most sub expressions and in direction of the outer most sub expressions (almost) as long as there are patterns that can be matched. In this example it could also match $F(X-1)-1$ to $X-1-1-1$. Drop the 1. from stack level 1, enter again $\{\&F(\&X)\ \ \&X-1\ \ TEST1\}$ and do another ↑ MATCH to get $X-1-1-1$ and see a 3 displayed on the top of the screen while the pattern matching is performed. Take a look what a cascade of events was caused by a single command execution. Purge now variable nIter, as we don't need it any more.

It could be interesting to see what the patterns $\&F(\&X)$, $\&X$, and $\&X-1$ were, each time the pattern matching is performed. We make the

$F(F(F(X)))$ $\{\&F(\&X)\ \ \&X-1\ \ TEST1\}$

↑ MATCH sees that the pattern $\&F(\&X)$ exists and triggers execution of TEST1

TEST1 runs for the first time. It initialises nIter to 0. It increments nIter to 1. It displays the current value of nIter on display line 1. It returns 1. (true)

$F(F(X-1))$

↑ MATCH sees that TEST1 evaluates to 1. (true) and so it performs pattern matching. After this it sees another pattern $\&F(\&X)$ and does the same again.

$F(F(X-1))\{\&F(\&X)\ \ \&X-1\ \ TEST1\}$

TEST1 runs for the second time. It increments nIter to 2. It displays the current value of nIter on display line 1. It returns 1. (true)

$F(X-1)-1$

↑ MATCH sees that TEST1 evaluates to 1. (true) and so it performs pattern matching. Then it stops though there is an additional (last) pattern $\&F(\&X)$.

program TEST4, which needs arguments. These are going to be exactly the patterns that we want to examine, after the pattern matching has been performed. Don't be confused about TEST4 coming directly after TEST1, since these programs are only examples. We could name them TESTWHATEVER, TESTASYOUWISH or anything else. The program TEST4 should create log records for each pattern matching in a nice readable form. So here we have TEST4:

```
<<
  IF
    VARS 'MATCHLOG' POS NOT
  THEN
    "" 'MATCHLOG' STO
  END
    'MATCHLOG' "Detected: F(X)=" 5. PICK +
    "
with: X=" 4. ROLL + ".
Replaced " + 4. ROLL + " with " + ROT +
"
--------------------------------
" + STO+ 1.
>>
```

Now enter a program that does pattern matching using the condition TEST(&F(&X),&X,&X−1) until nothing changes any more:

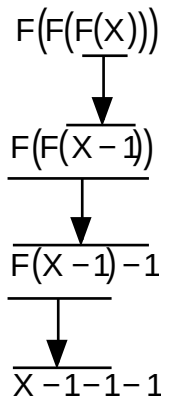
```
<<
  'F(F(F(X)))'
  DO
    {'&F(&X)' '&X-1' 'TEST4(&F(&X),&X,&X-1)' }
    ↑MATCH
  UNTIL
    NOT
  END
>>
```

Evaluate the program to get the fully matched result X−1−1−1. Press VAR to get the variables menu. Then press MATCHLOG and then ▼ to take a look at the log file. You see:

```
Detected: F(X)='F(X)'
with X=X.
Replaced 'F(X)' with 'X-1'
--------------------------------

Detected: F(X)='F(F(X-1))'
with X=F(X-1).
Replaced 'F(F(X-1))' with 'F(X-1)-1'
--------------------------------

Detected: F(X)='F(X-1)'
with X=X-1.
Replaced 'F(X-1)' with '(X-1)-1'
--------------------------------
```

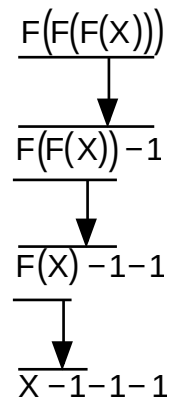


This tells you exactly how the pattern matching was performed. Purge variable MATCHLOG now, since we don't need it any more. Now, enter the same program as before, but using the command ↓MATCH instead of ↑MATCH. Let the program run. The result is again X−1−1−1, but it was produced in another way. Take a look at the contents of MATCHLOG. Now you see:

```
Detected: F(X)='F(F(F(X)))'
with X=F(F(X)).
Replaced 'F(F(F(X)))' with 'F(F(X))-1'
--------------------------------

Detected: F(X)='F(F(X))'
with X=F(X).
Replaced 'F(F(X))' with 'F(X)-1'
--------------------------------

Detected: F(X)='F(X)'
with X=X.
Replaced 'F(X)' with 'X-1'
--------------------------------
```



Purge again MATCHLOG.

In our example, when we successively match all patterns $F(X)$ with $X-1$, we end up with $X-1-1-1$, in which all information about where the expression came from is lost. Of course, in this case we know that each pattern $X-1$ came out of a pattern $F(X)$. But there are many cases in which losing the information about the expression before the match can be a big problem. You remember of course, what a problem it is, when expanding the derivative of a sum we just evaporate the variable of differentiation. But we can use the pattern matching commands with a condition that evaluates to true, and that creates all information for being able to reconstruct the original expression by doing the inverse pattern matching. Enter the program:

```
<<
  IF
    VARS 'RLIST' POS NOT
  THEN
    {} 'RLIST' STO
  END
  2. →LIST 1. →LIST 'RLIST' STO+
  1.
>>
```

and store it in TEST2. Now enter $F(F(F(X)))$ and then the list $\{\&F(\&X)\ \ \&X-1\ \ TEST2(\&F(\&X),\&X-1)\}$. Do a ↑ MATCH to get $F(X-1)-1$ and a 1. Drop the 1., enter the list $\{\&F(\&X)\ \ \&X-1\ \ TEST2(\&F(\&X),\&X-1)\}$ again and do another ↑ MATCH to get $X-1-1-1$ and a 1.. Drop the 1. from the stack. The variable RLIST was created by TEST2, the execution of which was triggered by the command ↑ MATCH. Recall RLIST on the stack, and take a look at its contents which are: $\{\{F(X-1)\ \ X-1-1\}\ \{F(F(X-1))\ \ F(X-1)-1\}\ \{F(X)\ \ X-1\}\}$. With this list we can do the reverse match and reconvert the result $X-1-1-1$ to what we started with. Enter a 1. and press ▶. The stack must contain now: On level 3 the algebraic $X-1-1-1$, on level 2 a 1., and on level 1 the list of the patterns for pattern matching $\{\{F(X-1)\ \ X-1-1\}\ \{F(F(X-1))\ \ F(X-1)-1\}\ \{F(X)\ \ X-1\}\}$. Enter the program:

```
<<
  REVLIST ↓MATCH DROP
>>
```

Press DOSUBS to get the original expression, $F(F(F(X)))$. The recorded patterns that were used to convert $F(F(F(X)))$ to $X-1-1-1$, were used again in reverse direction to convert $X-1-1-1$ to $F(F(F(X)))$. Purge now RLIST. This manual purging of the variables that are created is getting on our nervous. Can't it be done automatically? Of course! And there are many methods.

One of them would be to use local variables. Consider the program:

```
<<
  {} → rlist
  <<
    'F(F(F(X))'
    DO
      {'&F(&X)' '&X-1' 'TEST2(&F(&X),&X-1)' }
      ↓MATCH
    UNTIL
      NOT
    END
    rlist
  >>
>>
```

It would return the list of patterns and would create no global variable that must be explicitly purged. Another possibility is to use alarms. Let's

have a look at the programmable commands for alarms on the HP49G. First of all we have STOALARM. This command takes data from the stack and creates an alarm. It needs one argument, which can be:

1) A real number that specifies the time when the alarm will go off.
2) A list with two real numbers. The first specifies the date and the second specifies the time when the alarms will go off.
3) A list containing the date and the time as well the object that will be evaluated when the alarm will go off.
4) A list containing the date and the time, the object that will be evaluated when the alarm will go off, and a repeat interval for the alarm.

We will use the third variant, so let's see in more details what the contents of the list are. The first number is a real, which specifies the date of the alarm. It must be in format dd.mmyyyy or mm.ddyyyy accordingly to your setting of the date format. The second number is a real, which specifies the time of the alarm. It must be in format hh.mmss, with the part hh going from 0 to 24. The third element in the list can be any object. If it is a string, then we have an appointment alarm. At the specified date and time, the string is displayed and the HP49G beeps. But if it is anything else, a number, an expression, or a program, then we have a control alarm and the object is evaluated. STOALARM returns a number, the index of the alarm that it stores in the alarm list. The next command we examine is the command FINDALARM. It takes one argument from the stack, which can be:

1) A real number that specifies a date.
2) A list with two real numbers. The first specifies a date and the second specifies a time.
3) The number 0.

If the command is given a date, then it returns the index of the first alarm that comes due after 12:00 of the specified date. If it is given a list with a date and a time, then it returns the index of the first alarm that comes due after the specified date and time. If it is given a 0., then it returns the index of the first past due alarm. We also have the command DELALARM, which simply takes an alarm index an deletes the corresponding alarm from the alarm list.

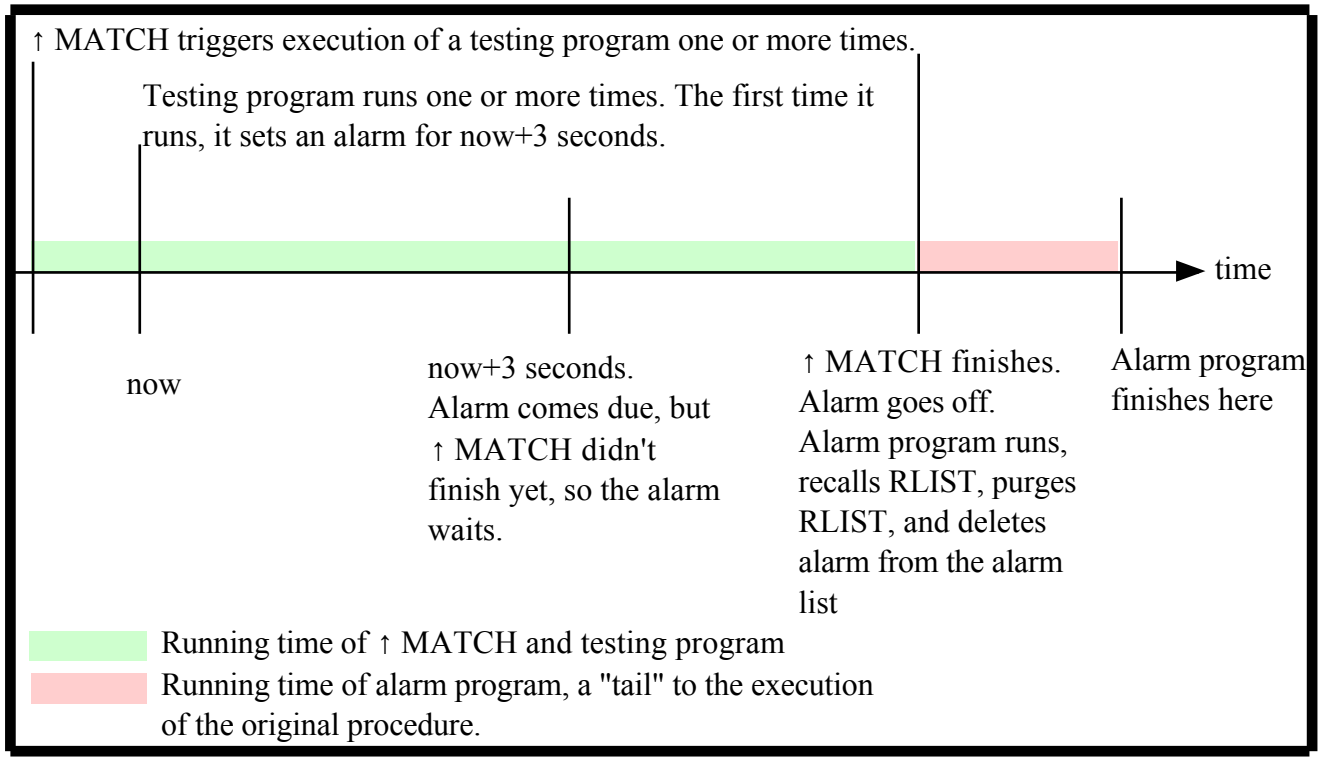
What we also should know about alarms on the HP49G is that they won't interrupt running programs. If an alarm comes due while a program is running, it will wait for the program to complete execution, and then it will go off. That means for us, that if we make the pattern matching condition program in such a way, that it sets an alarm to purge the variables created by the program, then it will not conflict with the program because it will come off after the program has completed. Let's try that. Enter the program:

```
<<
  IF                        @If RLIST doesn't exist
    VARS 'RLIST' POS NOT
  THEN                      @then initialise it
    {} 'RLIST' STO
    DATE TIME 0.0003        @Current date and time + 3s
    <<                      @Object to be executed
      RLIST 'RLIST' PURGE   @when the alarm comes due
      0. FINDALARM DELALARM
    >> 3. →LIST STOALARM DROP
  END
  2. →LIST 1. →LIST 'RLIST' STO+ 1.
>>
```

Store that in TEST3. Enter $F(F(F(X)))$ and then $\{\&F(\&X)\ \ \&X-1\ \ TEST2(\&F(\&X),\&X-1)\}$. Do a ↑ MATCH. The pattern matching is performed, and after that you get the variable RLIST recalled on the stack, and purged. The program

```
<<
  RLIST 'RLIST' PURGE
  0. FINDALARM DELALARM
>>
```

which runs when the pattern matching completes, also removes the alarm that triggered its execution from the the alarm list. Take a look at



the events as they take place as the time is passing by.
For our purposes the evaluation of the condition for pattern matching is a back door that enables us to enter the domain of "things that aren't possible". Consider for example some arbitrary expression containing derivatives of sums. In our programs we used pattern matching to prepare the expression so that only patterns of the form:

$$\frac{\partial}{\partial X}\left(\sum_{n=n0}^{N} F(X)\right)$$

exist. After this we want to convert all sums that appear inside the parentheses of $\frac{\partial}{\partial \text{var}}( )$ to temporary replacement functions. We want to have distinct replacement functions for distinct sums, and we want to retain he variable of differentiation, because if we don't, then expanding $\frac{\partial}{\partial \text{var}}(\text{tempFunction})$ would return 0, and the temporary replacement function would evaporate. We can make a program, name it for example CREATEMP, that we use as a condition for pattern matching. The next program listing demonstrates this.

```
<<
  IF                          @If RLIST doesn't exist
    VARS 'RLIST' POS NOT
  THEN                        @then initialise it
    {} 'RLIST' STO
    64. 'nTemp' STO           @initialise nTemp
  END
  "'tempFunction"             @Create tempFunction
  'nTemp' INCR CHR +
  "(" + SWAP +
  ")'" + OBJ->
  2. →LIST 1. →LIST
  'RLIST' STO+ 1.             @Add to replacement list
>>                            @and return 1. (true)
```

Store this in CREATEMP. Now, enter the expression:

$$\frac{\partial}{\partial X}\left(\sum_{n=a}^{b} G(X,n)\right)+\frac{\partial}{\partial X}\left(\sum_{n=1}^{N} F(X,n)^3\right)$$

that contains two different sums. Enter the pattern list:

$$\left\{\frac{\partial}{\partial \&V}\left(\sum_{\&n=\&n0}^{\&N} \&F\right)\ \ \frac{\partial}{\partial \&V}\left(\sum_{\&n=\&n0}^{\&N} \&F\right)\ \ \text{CREATEMP}\left(\sum_{\&n=\&n0}^{\&N} \&F,\&V\right)\right\}$$

and do a ↑ MATCH. When the pattern matching is done, you have exactly the same expression like before. But now you have the variable RLIST in the current directory, which contains sub lists with all the pairs of distinct sums and distinct temporary replacement functions. Now these pairs can be used for pattern matching. Remember that we did almost the same a few pages ago, when we used the program REPLΣ as a condition for pattern matching. But the difference was that we created an expression in which we had unevaluated patterns of the form:

$$\frac{\partial}{\partial X}\left(\text{tempFunctionA}(X)\right)$$

But now we just use pattern matching to trigger execution of the condition program, which in turn creates the list of replacements. The pattern matching command replaces each sum with itself, leaving the expression unchanged. The real replacement is to be done in a subsequent pattern matching operation using the created replacement list RLIST. Take a look its contents:

$$\left\{\left\{\sum_{n=1}^{N} F(X,n)^3\ \ \text{tempFunctionB}(X)\right\}\ \left\{\sum_{n=a}^{b} G(X,n)\ \ \text{tempFunctionA}(X)\right\}\right\}$$

It has all necessary information for replacement of sums with temporary functions that depend on the differentiation variable. Notice also how we use nTemp as a counter for adding a single capital letter to the string "'tempFunction", for creating *distinct* replacements for *distinct* sums. This method limits the number of possible distinct temporary replacement functions to 26, but even if this shouldn't be enough, we can improve it adding another letter, or finding some other method for creating distinct names of temporary replacement functions. Also, using the names tempFunctionA, tempFunctionB, and so on, means that these names must not appear in the original expression. If you for some reason have them in your expression, you could use replacement names like TrabakoulasA, or even KaragiaouroglouΣ. (And hope that the HP49G will not crash. ;-))

We stay a little bit longer at pattern matching. In the marathons (under the heavy influence of VPN ;-)) we often encountered a major shortcoming of the CAS of the HP49G. It has no INTEGERASSUME, and so it can't simplify for example SIN(n π) to 0, when n is integer. But does it really have no integer assuming capabilities? I would say that using pattern matching with conditions we can not only construct integer assumptions but also odd and even integer assumptions, or any other assumptions we want. Suppose that you want to expand the expression SIN(n π)+COS(n π) using integer assumptions for variable n. First of all go to directory CASDIR and store there the list {n} in variable INTEGERASSUME. Store the list {n} in variable ODDASSUME. Now enter the program:

```
<<
  {HOME CASDIR INTEGERASSUME} RCL SWAP POS
>>
```

Store it in HOME in variable ISINTEG?. Enter

```
<<
  {HOME CASDIR ODDASSUME} RCL SWAP POS
>>
```

Store it in HOME in variable ISODD? Now, suppose that you want to have a command, name it IEXPAND, that does expanding for trigonometrics using assumptions for integers and odd integers. Enter the program:

```
<<
  TEXPAND
  DO
    {'SIN(&n*π)' 0 'ISINTEG?(&n)'}
    ↑MATCH SWAP
    {'COS(&n*π)' -1 'ISODD?(&n)'}
    ↑MATCH ROT OR
  UNTIL
    NOT
  END
  EXPAND
>>
```

Store it in HOME in variable IEXPAND. Now return to your working directory. Since ISINTEG?, ISODD? and IEXPAND are in HOME, they are accessible from every directory in your directory structure. Enter $SIN(n\ \pi)+COS(n\ \pi)$. Enter IEXPAND and watch your HP49G using its new created knowledge to return -1. Imagine now, how many assumptions are possible using this simple method.

Another shortcoming of the HP49G is that the upwards pattern matching command ↑MATCH does (almost) all possible matches at any level of nesting, starting at the most inner nested sub expressions and making its way to the outer nestings. But using conditions we can make a new command, say ↑MATCH1, that does a single pattern matching at the inner most nested sub expression. Store the mini program

```
<<
  1. FS?C
>>
```

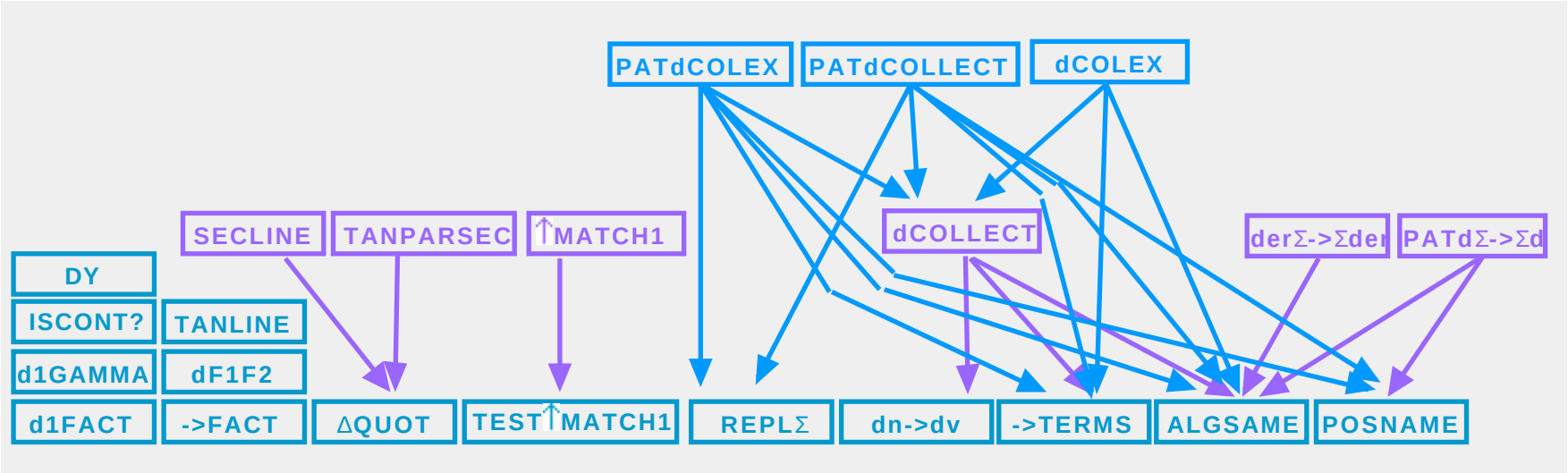
in TEST ↑ MATCH1. Store the program

```
<<
  PUSH 1. SF
  'TEST↑MATCH1' +
  ↑MATCH POP
>>
```

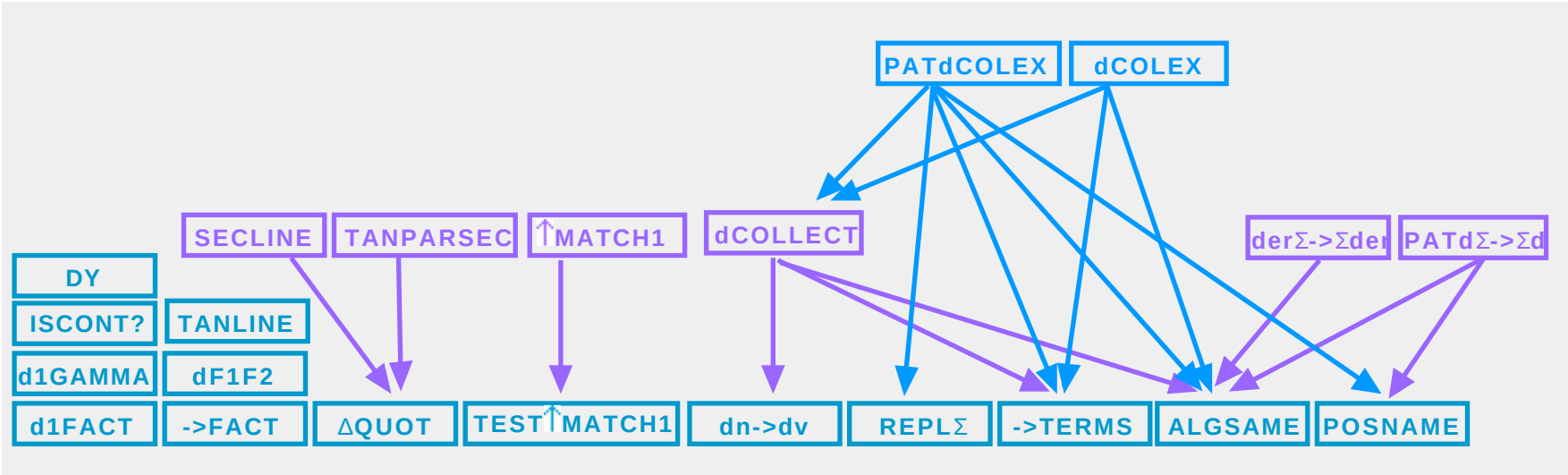
in ↑MATCH1. Now enter $F(F(F(X)))$, and then the pattern list $\{\&F(\&X)\ \ \&X-1\}$. Press TEST↑MATCH1 to get $F(F(X-1))$ and 1.. The expression $F(F(X-1))$ shows that pattern matching was performed only at the innermost nested sub expression.

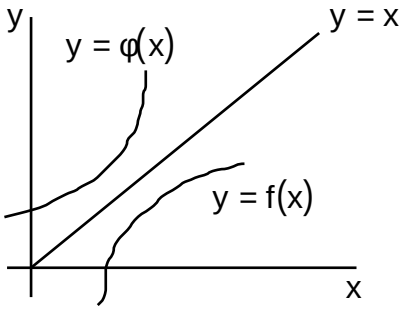
Enough patterns (for this time ;-)). In the next part of this marathon we are going to examine how to… put pattern matching in pattern matching. (Well, it seems that Nick has a preoccupation with such things ;-)) We are going to see, what would happen if we use some condition program for pattern matching that itself does pattern matching. And what happens if the program does pattern matching using itself as a condition program?

Before we go any further, let's take a look once more at our collection of programs, which has grown again (first picture on next page). I left out STARTEQW since it can use any other program, which means that its dependence on other programs will vary. If we through out PATdCOLLECT, whose functionality is implemented in PATdCOLEX, then the whole building gets a bit clearer (second picture on next page). The programs TEST1, TEST2, TEST3, TEST4 and CREATEMP are also left out, because they were introduced as examples for a possible usage of the pattern matching commands, but otherwise they aren't needed by any of our main programs. Notice that dCOLEX and PATdCOLEX (should) have the same functionality. Nonetheless I guess that there will be cases where the one works and the other doesn't work or even crashes. Decide for yourself which of them you find easier to understand and to make better.

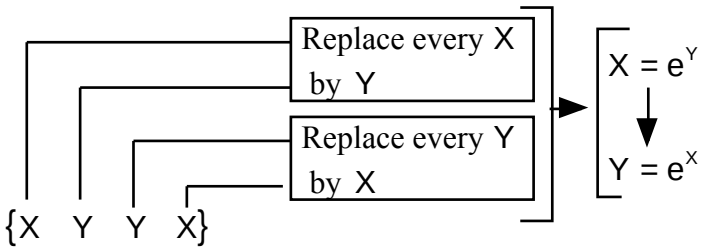


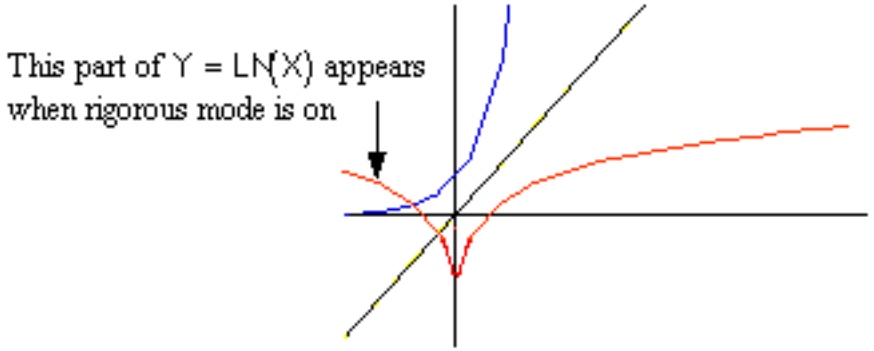
Anyway, it looks quite crowded in the neighbourhood of dn → dv, ALGSAME, POSNAME, → TERMS, and dCOLLECT. This shows that the functionality of these programs is essential. They are the fundament upon which the rest of the building stands. If they fall, all programs which base upon them will also fall.



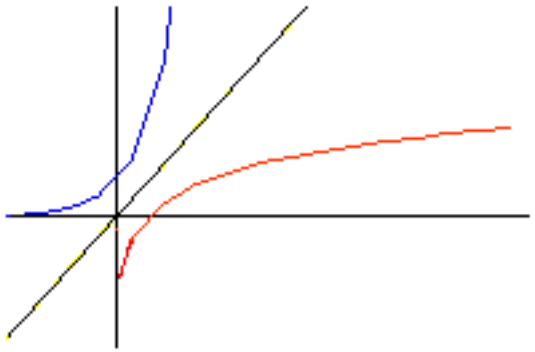


We come now to derivatives of inverse functions. The inverse function $y = \varphi(x)$ of a monotonic and continuous function $y = f(x)$ in the interval $a < x < b$ can be constructed geometrically, if we mirror the plot of the function on the line $y = x$. Analytically the function $y = \varphi(x)$ is found if we solve $y = f(x)$ for x, and exchange variables y and x in the solution. On the HP49G it is much easier to work analytically than geometrically for finding inverses. (And in general this is also *the* way to do this, as it provides us the equation of the inverse function, which we can use for potting and other purposes. For example, let's suppose that we have $Y = LN(X)$ and we want the inverse function. Go to the EQW, enter $Y = LN(X)$, and put that on the stack. Press ENTER to copy the equation on stack level 2. Enter X and press SOLVE, to get $X = e^Y$. Now we must exchange variables Y and X to get $Y = e^X$. If we use SUBST here, then we will have a problem. For example, if we enter $X = Y$ and press SUBST, then the result will be $Y = e^y$ and we will not be able to replace Y with X *only* in the exponential function, because if we enter $Y = X$ and substitute again, then we will get $X = e^X$ (i.e. *both* Ys will be substituted with Xs). But we can do what we want using |. Though in algebraic syntax the function | can perform *one* replacement at a time, in RPL syntax it can perform several replacements at once. If you enter {X Y Y X} and press |, then the two replacements will be performed simultaneously on the *original expression*. The result is then $Y = e^X$. In RPL syntax the command | gets one expression from stack level 2, and one list from stack level 1. The list contains in pairs the name that has to be replaced, and the expression that it has to be replaced with. The important thing here is that the replacements do not interfere with each other, even if some variable is replaced with another variable that already exists in the original expression and that itself has to be replaced with something else. Using | with the list {X Y Y X} on the HP49G is *not* equivalent to using SUBST once with $X = Y$ and right after this another time with $Y = X$! Now we have $Y = LN(X)$ and $Y = e^X$, that is the function and its inverse on the stack. Enter $Y = X$, the line on which we must mirror some function, in order to get its inverse. Now, enter 3 and press →LIST to make a list of all equations. Press STEQ to store the list of equations in EQ. Set the plot type to Function, independent variable to X, horizontal view from -7.5 to 12.5, vertical view from -4.2 to 5.8, and press ERASE and DRAW to plot the three functions. (If you have the Rcobo's HP49G with laser plasma screen, then you will see the plots in colour ;-)). But if rigourous mode is on, you are going to get an additional part of $Y = LN(X)$ which shouldn't be there. Exit the plot, return to the stack, enter X, and press LN. In rigourous mode the result is $LN(|X|)$, which explains how the





additional part in the plot was drawn. The HP49G didn't actually plot $Y = LN(X)$, but rather $Y = LN(|X|)$, because the CAS was in rigourous mode. Here we have another (unneeded) complication. While in general it is good to work in rigourous more (i.e. not to assume that $|X| = X$), for the plot of $LN(X)$ this brings an additional thing that we must do. Press MODE, then the menu key CAS, and then deactivate the option _Rigorous. Press ENTER to accept the changes, then ENTER again to leave the screen CALCULATOR MODES, and finally press ERASE and DRAW again, to re-plot. Now you get the correct plot without the additional part.



If we have the inverse function $x = \varphi(y)$ of some function $y = f(x)$, but not the function $y = f(x)$ itself, and we want the derivative of $y = f(x)$, then we don't need to find the function $y = f(x)$ first. Instead of this we can use the relation:

$$\frac{\partial f}{\partial x} = \frac{1}{\frac{\partial \varphi}{\partial y}}$$

Which on the HP49G is piece of cake. We do a simple example first. We use $X(Y) = LN(Y)$ which is the inverse of $Y(X) = e^X$. In this example it is easy to find the function out of its inverse, but this doesn't always have to be this case, as we will see in the next example. Enter now $LN(Y)$, then Y, and then press ∂ or DERIV to get $\frac{1}{Y}$. Press 1/X to get Y. The derivative of the original function $Y = e^X$ is Y. Notice that we find the derivative in terms of Y and not in terms of X. Indeed the derivative of $Y(X) = e^X$ is:

$$\frac{\partial Y(X)}{\partial X} = \frac{\partial e^X}{\partial X} = e^X = Y(X)$$

We continue with an example in which it isn't possible to find the analytic closed form of the original function. We use as the inverse function $X(Y) = Y\ e^Y$. Enter $Y\ e^Y$, then Y, and then press ∂ or DERIV to get $e^Y + Y\ e^Y$. Press 1/X to get:

$$\frac{1}{e^Y + Y\ e^Y}$$

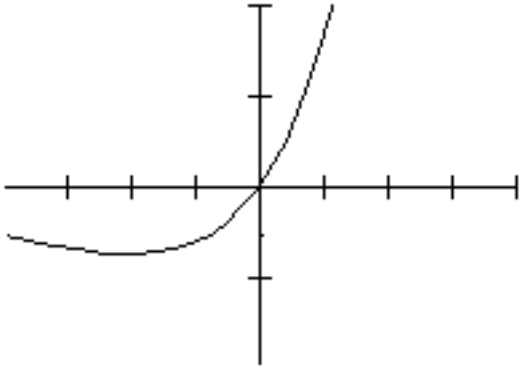
which is the derivative of the original function $Y(X)$, in terms of Y. The function $Y(X)$ itself that can't be written in an analytic closed form in terms of X. But nonetheless we have found out that:

$$\frac{\partial Y(X)}{\partial X} = \frac{1}{e^{Y(X)} + Y(X)\ e^{Y(X)}}$$

i.e. the derivative of $Y(X)$ in terms of $Y(X)$.

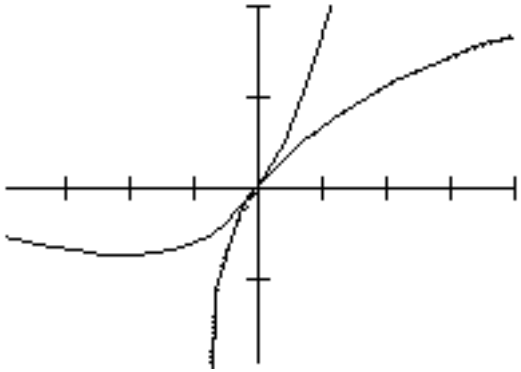
An additional problem that we have when we know the inverse function but we can't find the original function, is how to plot the original function. In our example from above, $X = Y\ e^Y$ we can plot of course $Y\ e^Y$, but this will be the picture of the inverse function, and we must imagine what it would look like, if we mirrored the function curve on the line $Y = X$. Let's see how we can do that without having to imagine mirror worlds. Set up a function plot with $Y\ e^Y$ as EQ and Y as indep. Set horizontal view range from -2 to 2, and vertical view range

from -1 to 1. Plot the function to get a curve like in the picture to the right. Now set up a parametric plot with $Y\ e^Y + i\ Y$ as EQ. The equation of the parametric function for plotting the original equation is easily found. Since we want to have $X = Y\ e^Y$, we write this in parametric form using Y itself as parameter:



$$X = Y\ e^Y$$
$$Y = Y$$

This parameter representation, translated in HP49G-ish, is the same as $Y\ e^Y + i\ Y$. Now the plot contains both the original function, of which we can't have an analytic closed form, and the inverse function. This technique can be used for visualising the curve of the original function, when we can't represent it by means of an algebraic equation of the form $y = f(x)$. Notice that in many cases the original "function" is actually no function at all, but rather a relation. The above example is one of these cases.



While we are talking about functions in parameter representation, the strange derivatives of the form $\frac{\partial f(x)}{\partial g(x)}$ come into mind, which we already encountered. Let's take a look at parametric functions and their derivatives. A function $y = f(x)$ is given in parametric form:

$$y = \phi(t)$$
$$x = \psi(t)$$

We don't know how y depends on x, but we know how x and y both depend on the parameter t. From the parametric representation we can find the derivative $\frac{\partial y(x)}{\partial x}$ without having to find $y(x)$ itself. For the derivative $\frac{\partial y(x)}{\partial x}$ we have:

$$\frac{\partial y(x)}{\partial x} = \frac{\frac{\partial y(t)}{\partial t}}{\frac{\partial x(t)}{\partial t}}$$

That means that we can find the derivative $\frac{\partial y(x)}{\partial x}$ using the derivatives $\frac{\partial y(t)}{\partial t}$ and $\frac{\partial x(t)}{\partial t}$. Let's have an example. The equation of an ellipse:

$$\frac{X^2}{A} + \frac{Y^2}{B} = 1$$

in parametric form is:

$X = A\ \mathrm{COS}(t)$

$Y = B\ \mathrm{SIN}(t)$

Now we find the derivative. Enter $B\ \mathrm{SIN}(t)$, then t, and press ∂ or DERIV to get $B\ \mathrm{COS}(t)$. This is the derivative:

$$\frac{\partial y(t)}{\partial t}$$

Now, enter $A\ \mathrm{COS}(t)$, then t, and press ∂ or DERIV to get $A\ -\mathrm{SIN}(t)$. This is the derivative:

$$\frac{\partial x(t)}{\partial t}$$

Press ÷ and then EXPAND to get:

$$-\frac{B\ \mathrm{COS}(t)}{A\ \mathrm{SIN}(t)}$$

This is the derivative:

$$\frac{\partial y(x)}{\partial x}$$

written in terms of t. But of course we can write it also as a function of X, or of Y, or of both X and Y, if we want. For example, since $X = A\ \mathrm{COS}(t)$, we have:

$$\mathrm{COS}(t) = \frac{X}{A}$$

And since $Y = B\ \mathrm{SIN}(t)$, we have:

$$\mathrm{SIN}(t) = \frac{Y}{B}$$

How to substitute

$$\mathrm{COS}(t) = \frac{X}{A}$$

and

$$\mathrm{SIN}(t) = \frac{Y}{B}$$

in

$$-\frac{B\ \mathrm{COS}(t)}{A\ \mathrm{SIN}(t)}\ ?$$

Let's do that all on the HP49G. Press ENTER a couple of times to make several copies of:

$$-\frac{B\ \mathrm{COS}(t)}{A\ \mathrm{SIN}(t)}$$

on the stack, because we are going to need the expression more than one times. Enter $X = A\ \mathrm{COS}(t)$ and then $\mathrm{COS}(t)$. Press SOLVE to solve for $\mathrm{COS}(t)$. As we already saw, the command SOLVE allows to solve for any *rational* variable of a given equation. The result is:

$$\mathrm{COS}(t) = \frac{X}{A}$$

Press SUBST, and you will not get:

$$-\frac{B\ \frac{X}{A}}{A\ \mathrm{SIN}(t)}$$

but rather:

$$-\frac{B\ \frac{X}{A}}{A\ \mathrm{SIN}\left(\mathrm{ACOS}\left(\frac{X}{A}\right)\right)}$$

How and why is this result obtained? The command SUBST didn't really substitute $\frac{X}{A}$ for all $\mathrm{COS}(t)$ that it found. In fact it didn't even search for $\mathrm{COS}(t)$. What it did seems to be:

1) Solve the equation $\mathrm{COS}(t)=\frac{X}{A}$ for the first variable (not *rational* variable) on the left hand side, which is t. This returns $t=\mathrm{ACOS}\left(\frac{X}{A}\right)$. Notice that this is not the general solution that SOLVE would return.

2) Substitute $t=\mathrm{ACOS}\left(\frac{X}{A}\right)$ in $-\frac{B\ \mathrm{COS}(t)}{A\ \mathrm{SIN}(t)}$. This returned the result $-\frac{B\ \frac{X}{A}}{A\ \mathrm{SIN}\left(\mathrm{ACOS}\left(\frac{X}{A}\right)\right)}$.

In the first step, the solution of $\mathrm{COS}(t)=\frac{X}{A}$ for t, was found to be:

$$t=\mathrm{ACOS}\left(\frac{X}{A}\right)$$

This is the principal solution of the equation. The general solution that SOLVE would return, would be:

$$\begin{cases} t=-\left(2\ n1\ \pi+\mathrm{ACOS}\left(\frac{X}{A}\right)\right) \\ t=2\ n1\ \pi+\mathrm{ACOS}\left(\frac{X}{A}\right) \end{cases}$$

The same solution is returned by ISOL, when flag -1 is clear. But when flag -1 is set, then ISOL returns the principal solution:

$$t=\mathrm{ACOS}\left(\frac{X}{A}\right)$$

SUBST returns always the principal solution. You don't believe that SUBST can be used for this? Very well! Enter:

$$\mathrm{COS}(t)=\frac{X}{A}$$

then t, and press SUBST to get:

$t=\mathrm{ACOS}\left(\frac{X}{A}\right)$. When SUBST is used with an equation on stack level 1, it does substitution. When it is used with an expression or name (no equation) on stack level 1, it solves the expression on stack level 2 for the expression or name on stack level 1. So we have:

	Flag -1 clear for general solutions	Flag -1 set for principal solutions
SOLVE	SOLVE doesn't care about flag -1 and always returns general solutions.	
ISOL	General solution.	Principal solution.
SUBST	SUBST doesn't care about flag -1 and always returns principal solutions.	

Notice how wonderful the three commands cover the whole spectrum of possibilities.

Now, let's go on with our problem. In the expression:

$$-\frac{B\ \frac{X}{A}}{A\ SIN\left(ACOS\left(\frac{X}{A}\right)\right)}$$

we can't do the second substitution $SIN(t)=\frac{Y}{B}$, because the variable t doesn't exist anymore. We can only expand the expression to get:

$$-\frac{X\ B\ \sqrt{A^2-X^2}\ |A|}{A^4-X^2\ A^2}$$

This is the derivative $\frac{\partial y(x)}{\partial x}$, written as a function of X. But we wanted it as a function of X and Y. Drop the expression:

$$-\frac{X\ B\ \sqrt{A^2-X^2}\ |A|}{A^4-X^2\ A^2}$$

from the stack and let's start over. What we need is to replace $COS(t)$ with $\frac{X}{A}$, and $SIN(t)$ with $\frac{Y}{B}$ in:

$$-\frac{B\ COS(t)}{A\ SIN(t)}$$

without doing anything else. This is clearly a mission for pattern matching. With:

$-\frac{B\ COS(t)}{A\ SIN(t)}$ on stack level 1, enter the list:

$$\left\{\ COS(t)\ \ \frac{X}{A}\ \right\}$$

and press ↑MATCH, to get:

$$-\frac{B\ \frac{X}{A}}{A\ SIN(t)}$$

and a 1. Drop the 1., enter:

$$\left\{\ SIN(t)\ \ \frac{Y}{A}\ \right\}$$

and press ↑MATCH a second time, to get:

$$-\frac{B\ \frac{X}{A}}{A\ \frac{Y}{B}}$$

and a 1. Drop the 1. and expand to get:

$$-\frac{X\ B^2}{Y\ A^2}$$

the derivative as a function of X and Y.

Obtaining the derivative as function of Y alone is also easy. Drop the expression:

$$-\frac{X\ B^2}{Y\ A^2}$$

Now the expression:

$$-\frac{B\ COS(t)}{A\ SIN(t)}$$

should be on stack level 1. Enter $Y = B\ SIN(t)$, the definition of Y as a function of the parameter t. Enter t, and press SUBST, to get the principal solution:

$$t = ASIN\left(\frac{Y}{B}\right)$$

Press SUBST again to get:

$$-\frac{B\ COS\left(ASIN\left(\frac{Y}{B}\right)\right)}{A\ SIN\left(ASIN\left(\frac{Y}{B}\right)\right)}$$

Expand the expression to get:

$$-\frac{B^2\ \sqrt{B^2-Y^2}\ |B|}{SQ(B)\ Y\ A}$$

This is not completely expanded, so press EXPAND again. You get:

$$-\frac{\sqrt{B^2-Y^2}\ |B|}{Y\ A}$$

This is the derivative as a function of Y alone.

We see from the above that when we have a function $y(x)$ written in parametric form:

$$x = \phi(t)$$
$$y = \psi(t)$$

it is easy to get the derivative $\frac{\partial y(x)}{\partial x}$ as a function of the parameter t. We only need to calculate:

$$\frac{\frac{\partial y(t)}{\partial t}}{\frac{\partial x(t)}{\partial t}}$$

The problem is how to convert it to a function of x, or of y, or of x and y. The conversion to a function of x or of y alone is systematically easier. We just solve the equation of the parametric definition of x of or y for the parameter t. Then we substitute the solution for t in the expression:

$$\frac{\frac{\partial y(t)}{\partial t}}{\frac{\partial x(t)}{\partial t}}$$

The difficulties here arise from the fact that it will not always be possible to solve the parametric definition of x or y for the parameter t. But this is not a systematical difficulty. The procedure is clear and very easily to implement on a machine. On the other hand, turning

$$\frac{\frac{\partial y(t)}{\partial t}}{\frac{\partial x(t)}{\partial t}}$$

to a function of x and y by "seeing" what patterns are best available for matching, is for us humans an easy thing. But how can we do that systematically, so that we can show the poor HP49G what it should do? There is no general recipe for "seeing", like we do. For pattern recognition we, humans, are still unbeatable. We can recognise patterns by just taking a look at them. As Trabakoulas says, "…this is our strength. We can recognise a face instantly, without the need to calculate angles between the nose and the eyes, and distances from one ear to the other. We just see that. The HP49G can't see that. But that might be also our weakness. We sometimes "see" to much, without really taking care to derive truth of falseness of what we "see". In this category of phenomena belongs the old picture of automatically "knowing who the bad guys and who the good guys are". Just because we "saw" what kind of clothes they have, what they believe to be God, or what their opinions about this world are. No matter what we say about our progress in these things, the old devil of pattern recognition is in our minds since the first humans walked on this planet. This is the way biosystems work. If we had to calculate first, if the beautiful animal that comes running to us, is a lion, we would presumably not have any need for discussing about capabilities of CAS, simply because we would be eaten out, long before getting the idea to make a CAS. Of course it is not impossible that some of these lions would suddenly decide to not eat us, but the pattern recognition machine says that this is rather unlikely - almost impossible - and so it puts the 2 hypothetical lions that wouldn't eat us in the same category of lions that would eat us. It is a safe method, good for surviving. But think about the lost possibilities. A powerful friend, the vegetarian lion, is lost right from the start. If we are to make real progress in these things, if we want to face the world without prejudice and ad hoc categories, we have to diminish the importance of the pattern recognition unit in our brains. To control it with logic, when it is telling us something about a pattern. And we can do that only after accepting that the pattern recognition unit it still is there, it still works, it still produces patterns - the fundaments of our amazing capabilities, and also the fundament of our too fast categorising everything." Enough philosophy, let's continue our marathon.

Let's make a program for finding the derivative of a function in parametric form. The program should return the derivative as a function of the parameter, as a function of the independent variable, and as a function of the dependent variable. It should also try to give us the derivative as a function of the independent and the dependent variable. Of course the latter will be rather imperfect, but at least we can try to imitate our built-in pattern recognition unit on the HP49G. This naive imitation is based on the above example. The program finds the rational variables of the definitions of x and y as functions of the parameter. It picks the first rational variable in each definition, solves for this variable, and tries to replace this rational variable with the solution in the derivative. In the above example, the derivative as a function of the parameter is:

$$-\frac{B\ COS(t)}{A\ SIN(t)}$$

The definitions for x and y are: $X = A\ COS(t)$ and $Y = B\ SIN(t)$. The rational variables that appear in the first definition are $COS(t)$ and A. The program solves $X = A\ COS(t)$ for $COS(t)$ and finds:

$$COS(t) = \frac{X}{A}$$

Then it matches $COS(t)$ to $\frac{X}{A}$ in the derivative:

$$-\frac{B\ COS(t)}{A\ SIN(t)}$$

and finds:

$$-\frac{B\ \frac{X}{A}}{A\ SIN(t)}$$

The same it does for the other definition, $Y = B\ SIN(t)$. (Very naive, I know, but perhaps somebody is going to find out how we do it, in our brains ;-)) Since the representation of a function in parametric form (for plotting) is $X + i\ Y$, we retain this syntax. The program will take four arguments from the stack. The parametric function, the X-variable, the Y-variable and the parameter. It will return the representations of the derivative, which we already examined. That is, it will return the derivative as a function of the parameter, as a function of the X-variable, as a function of the Y-variable and as a function of the X and the Y variable. All outputs will be labelled.

```
<<
  → paramFunc var1 var2 param
  <<
    PUSH {1. 2.} CF        @Flags 1 and 2 are used
    paramFunc RE
    paramFunc IM
    2. →LIST DUP param ∂   @Find der.
    OBJ→ DROP SWAP /
    EXPAND DUP
    "der(" param + ")" +   @Label der. as function of
     →TAG                  @the parameter
    OVER 4. PICK HEAD var1 @Try to solve x=ϕ(t) for t
    = param
    IFERR                  @If error during solving
      SOLVE
    THEN                   @then wrap der., x=ϕ(t) and t
      3. →LIST "Error"     @in a list and label it
       →TAG                @with "Error".
    ELSE                   @Else (no error during SOLVE)
      IF                   @If
        DUP {} SAME        @no solutions found
      THEN                 @then
        DROP               @drop the empty list
        "No sol. for "     @Make label for no solution
        var1 +
      ELSE                 @Else (we have solutions)
        SUBST              @substitute them in deriv.
        "der(" var1 + ")"  @Make label
        +
      END
    END
     →TAG                  @Label result
    PICK3 5. PICK 2. GET
    var2 = param           @Try to solve y=ψ(t) for t
    IFERR                  @If error during solving
      SOLVE
    THEN                   @then wrap der., x=ϕ(t) and t
      3. →LIST "Error"     @in a list and label it
       →TAG                @with "Error".
    ELSE                   @Else (no error during SOLVE)
      IF                   @If
```

```
    DUP {} SAME          @no solutions found
  THEN                   @then
    DROP                 @drop the empty list
    "No sol. for "       @Make label for no solution
    var2 +
  ELSE                   @Else (we have solutions)
    SUBST                @substitute them in deriv.
    "der(" var2 + ")"    @Make label
    +
  END
END
 → TAG
4. ROLL 5. ROLL          @Here starts the naive code
OBJ→ DROP
 →  derParm x y
<<
  x LVAR                 @Find rational vars. of x
  1. OVER SIZE HEAD
  FOR I
    IF
      DUP I GET LNAME    @Find all names in rat. var.
      IF                 @Convert to list if
        DUP {} ≠         @necessary
      THEN
        AXL
      END
      param POS          @If param. in rat var.
    THEN                 @then we use this rat. var.
      9.99999999999E499
      'I' STO            @Store MAXR in I (to exit
      1. SF              @loop). Set flag 1.
      NIP SWAP var1 =    @Try to solve x=ϕ(t) for
      SWAP               @the 1st. rat. var. that
      IFERR              @contains the param.
        SOLVE            @In case of error
      THEN               @Return labelled list with
        2.  →LIST        @x=ϕ(t) and t.
        "Error"  →TAG
        2. SF            @and set flag 2.
      ELSE               @else (no err. during SOLVE)
        OBJ→  DROP       @make list for matching
        →LIST
        derParm SWAP
        ↑MATCH DROP
        'derParm' STO
      END
    ELSE                 @Else (param. not in rat.
      DROP               @var.) drop rat. var.
    END
NEXT
IF                       @If we didn't find a rat. var.
  1. FC?C                @that contains the param.
THEN                     @then
  DROP2                  @drop 2 objects.
END
IF                       @If no error while solving
  2. FC?C                @x=ϕ(t) for rat. var.
THEN                     @then we try to solve also
  y LVAR 1. OVER         @y=ψ(t) for a rat. var. that
  SIZE HEAD              @contains the parameter.
  FOR I                  @We do the same like for
    IF                   @x=ϕ(t)
      DUP I GET
      LNAME
      IF
        DUP {} ≠
      THEN
        AXL
      END
      param POS
    THEN
      9.99999999999E499
      'I' STO
      1. SF NIP
      SWAP var2 =
      SWAP
      IFERR
        SOLVE
      THEN
        2.  →LIST
        2.  →LIST
        "Error"  →TAG
```

```
            2. SF
          ELSE
            OBJ→ DROP
            →LIST derParm
            SWAP ↑MATCH DROP
          END
        ELSE
          DROP
        END
      NEXT
      IF
        1. FC?
      THEN
        DROP2
      END
      IF                  @If there was a rat. var.
        1. FS? 2. FC?     @that contained the param.
        AND               @and a solution was found
      THEN                @then expand and label
        EXPAND "der("
        var1 + "," var2 +
        ")" + →TAG
      END
    END
  >>
  POP
>>
>>
```

This is the program dPARMF that comes with this document. The coloured code is the naive part of the program. (That's why it is in baby blue ;-)) If you don't want to have it, rip it off out of the program, (Poor baby, away from mama ;-)) Let's test the program and see what it does. Enter the parametric function of our example from above as: $A\ COS(t)+i\ B\ SIN(t)$. Enter X (the independent variable), then Y (the dependent variable), and then t (the parameter). *Switch to real mode*. That's important!!! If you are in complex more, and some of the variables A, B, X, Y or t is not assumed to be real, then the HP49G will consider it as a complex quantity and will not five the results that are described in the next paragraphs. Press dPARMF and wait some seconds. When the program finishes, you have the following results:

On stack level 4 the derivative of the parametric function, written as a function of the parameter t:

$$der(t):\left(-\frac{B\ COS(t)}{A\ SIN(t)}\right).$$

On stack level 3 the derivative of the parametric function, written as a function of the independent variable X:

$$der(X):\left\{-\frac{B\ COS\left(\frac{\pi}{2}\ n1\ \pi+ACOS\left(\frac{X}{A}\right)\right)}{A\ SIN\left(\frac{\pi}{2}\ n1\ \pi+ACOS\left(\frac{X}{A}\right)\right)}\quad -\frac{B\ COS\left(\frac{\pi}{2}\ n1\ \pi+ACOS\left(\frac{X}{A}\right)\right)}{A\ SIN\left(\frac{\pi}{2}\ n1\ \pi+ACOS\left(\frac{X}{A}\right)\right)}\right\}$$

The program returns the general solution because the might be cases in which the principal solution is not what we want. Substituting $n1=0$, and expanding will give us the result:

$$-\frac{X\ B\ \sqrt{A^2-X^2}\ |A|}{A^4-X^2\ A^2}$$

that we had on page 2-79.

On stack level 2 the derivative of the parametric function, written as a function of the dependent variable Y:

$$der(Y):\left\{-\frac{B\ COS\left(\frac{\pi}{2}\ n1\ \pi+ASIN\left(\frac{Y}{B}\right)\right)}{A\ SIN\left(\frac{\pi}{2}\ n1\ \pi+ASIN\left(\frac{Y}{B}\right)\right)}\quad -\frac{B\ COS\left(\frac{\pi}{2}\ n1\ \pi+ASIN\left(\frac{Y}{B}\right)\right)}{A\ SIN\left(\frac{\pi}{2}\ n1\ \pi+ASIN\left(\frac{Y}{B}\right)\right)}\right\}$$

Again, the program returns the general solution because the might be cases in which we don't want the principal solution. Substituting with $n1=0$, and expanding will give us the result:

$$-\frac{\sqrt{B^2-Y^2}\ |B|}{Y\ A}$$

that we had on page 2-80.

On stack level 1 the derivative of the parametric function, written as a function of the independent variable X and the dependent variable Y:

$$\text{der}(X,Y):\left(-\frac{X\ B^2}{Y\ A^2}\right)$$

The naive part of the code, did its work OK in this case.

We try another example. The parametric representation of the epicycloid (on the HP49G) is $A\ (t-\text{SIN}(t))+i\ A\ (1-\text{COS}(t))$. enter this expression, and then X, Y, and t. (Again, make sure you are in real mode). Press dPARMF and wait. After some seconds you get:

$$\text{der}(t):\left(-\frac{\text{SIN}(t)}{A\ \text{COS}(t)-A}\right)$$

$$\text{Error:}\begin{pmatrix}-\frac{\text{SIN}(t)}{A\ \text{COS}(t)-A} & A\ (t-\text{SIN}(t))=X & t\end{pmatrix}$$

$$\text{der}(Y):\begin{pmatrix}-\frac{\text{SIN}(-((2\ n1+1)\ \pi-\text{ACOS}(Y-1)))}{A\ \text{COS}(-((2\ n1+1)\ \pi-\text{ACOS}(Y-1)))-A}\\ -\frac{\text{SIN}((2\ n1+1)\ \pi-\text{ACOS}(Y-1))}{A\ \text{COS}((2\ n1+1)\ \pi-\text{ACOS}(Y-1))-A}\end{pmatrix}$$

$$\text{der}(X,Y):\frac{t\ A-X}{Y\ A^2}$$

The result labelled with Error on stack level 3 shows that the HP49G couldn't solve $A\ (t-\text{SIN}(t))=X$ for t, and so it couldn't substitute $t=\text{someFunctionOf}(X)$ in:

$$-\frac{\text{SIN}(t)}{A\ \text{COS}(t)-A}$$

(But who can solve $A\ (t-\text{SIN}(t))=X$ for t analytically?)

The result on stack level 1 shows that the naive code already had problems. It couldn't convert the derivative to a function that depends on X and Y, but not on t. Oh well, we are just at the beginning of programming automatic pattern recognition and artificial intelligence. Except of course if somebody connects a brain directly to the HP49G. Then two things may happen:

1) The HP49G will be glad to have artificial intelligence. This is the case for most people out there.

2) The human will start beeping. (This is the case for Nick ;-))

Perhaps you already noticed that the program dF1F2 of the first part of this marathon is a relative of the program dPARMF. The program

dF1F2 finds derivatives of the form $\frac{\partial f(x)}{\partial g(x)}$. It uses the following technique to do its work.

First it sets $g(x) = ttemp$, introducing ttemp as a new variable. Then it (tries to) solve $g(x) = ttemp$ for x creating the solution $x = someFunctionOf(ttemp)$. (The function $someFunctionOf(ttemp)$ is the returned solution $x =$.) Then it substitutes $g(x) = ttemp$ and $x = someFunctionOf(ttemp)$ in $\frac{\partial f(x)}{\partial g(x)}$, creating:

$$\frac{\partial someFunctionOf(ttemp)}{\partial ttemp}$$

This derivative is evaluated returning the function $firstDerivativeOfsomeFunctionOf(ttemp)$. Then $ttemp = g(x)$ is substituted in this function, creating $firstDerivativeOfsomeFunctionOf(g(x))$, which is returned by dF1F2. The relation of the two programs is best seen in a table on the top right. For dF1F2 that means that we can also use the same mechanism as in dPARMF. We only have to do some re-constructing in order to create the parametric definition. For example, let's reconsider the derivative:

$$\frac{\partial SIN(X)}{\partial COS(X)}$$

that we had to do with on page 1-24. If we consider X as a parameter, then we can write:

Program / Properties	dPARMF	dF1F2
Definition of the function that is differentiated	$\begin{cases} y = \psi(t) \\ x = \phi(t) \end{cases}$ $y(x) = y(\phi(t))$	$g(x) = ttemp$ $x = someFunctionOf(ttemp)$ $f(x) = f(someFunctionOf(ttemp))$
Explanations	Consider the definitions as $\psi(t) = y$, $\phi(t) = x$, rename t to x, and x to ttemp, and you have the same situation as for dF1F2.	Consider the definitions as $y = \psi(x)$, $ttemp = \phi(x)$, rename ttemp to x, and x to t, and you have the same situation as for dPARMF.

$$\begin{cases} y = SIN(X) \\ x = COS(X) \end{cases}$$

(The variables x and X are *not* the same.) To find the derivative:

$$\frac{\partial SIN(X)}{\partial COS(X)}$$

we simply have to find:

$$\frac{\frac{\partial SIN(X)}{\partial X}}{\frac{\partial COS(X)}{\partial X}}$$

Ha! That means that dF1F2 can be rewritten simpler and shorter.

```
<<
  → y x param
  <<
    y param ∂ x param ∂
    / EXPAND
  >>
>>
```

The program in its new version requires an additional argument, namely the name for the parameter on which y and x depend. Generally speaking it is a good policy to make programs that must have the name for which we do something (in our case differentiation). We could have to find:

$$\frac{\partial \text{SIN}(X\ A)\ e^{Z}}{\partial A\ Z\ \text{COS}(X)}$$

or any other derivative in which more than one variables are involved, and so we must know what to consider as the parameter for which we differentiate. Before we store the new program in dF1F2, we test the old version of dF1F2. Let's use the above example. Enter $\text{SIN}(X\ A)\ e^{Z}$, then $A\ Z\ \text{COS}(X)$, and press dF1F2. The program returns:

$$\frac{X\ e^{z}\ \text{COS}(X\ A)}{Z\ \text{COS}(X)}$$

which is correct… only if the variable A was meant as the parameter. The program in its old version automatically assumes the first name in the vector of names to be the parameter. Type in the small program on the top of this column and store it in dF1F2. Enter $\text{SIN}(X\ A)\ e^{Z}$, $A\ Z\ \text{COS}(X)$ and X. Press dF1F2 to get:

$$-\frac{e^{z}\ \text{COS}(X\ A)}{Z\ \text{SIN}(X)}$$

the correct result if we consider X as the parameter. The result that the old version returned, considering A as the parameter, can be obtained by the new version, if you enter $\text{SIN}(X\ A)\ e^{Z}$, $A\ Z\ \text{COS}(X)$ and A, and press dF1F2. The program dPARMF does the work of dF1F2 but it also does additional work, trying to convert the derivative to expressions that depend on the variables instead on the parameter. We can use it to find:

$$\frac{\partial \text{SIN}(X\ A)\ e^{Z}}{\partial A\ Z\ \text{COS}(X)}$$

We enter $A\ Z\ \text{COS}(X)+i\ \text{SIN}(X\ A)\ e^{Z}$, x, y, X, and press dPARMF. Notice that in this example the variables are the small letters x and y, and the parameter is the capital letter X. The program returns the result:

$$\text{der}(X):\begin{pmatrix} -\dfrac{e^{z}\ \text{COS}(X\ A)}{Z\ \text{SIN}(X)} \end{pmatrix}$$

on stack level 4, and the other results on stack levels 3 to 1.

Before we proceed we notice that the differentiation commands work also with lists. The following picture on the next page demonstrates how these commands behave when one or both arguments are lists.

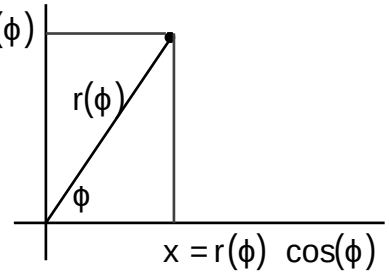
$$\{f(x)\ \ g(x)\ \}\ \xrightarrow{\partial \text{ or DERIV}}\ \begin{bmatrix}\frac{\partial f(x)}{\partial x} & \frac{\partial g(x)}{\partial x}\end{bmatrix}$$
$$x$$

$$f(x,y,\ )\ \xrightarrow{\partial \text{ or DERIV}}\ \begin{bmatrix}\frac{\partial f(x,y,\ )}{\partial x} & \frac{\partial f(x,y,\ )}{\partial y}\end{bmatrix}$$
$$\{x\ \ y\ \}$$

$$\{f(x)\ \ g(y)\ \}\ \xrightarrow{\partial \text{ or DERIV}}\ \begin{bmatrix}\frac{\partial f(x)}{\partial x} & \frac{\partial g(y)}{\partial y}\end{bmatrix}$$
$$\{x\ \ y\ \}$$

Now that we have examined derivatives of parametric functions, we can proceed to derivatives of functions in polar form. A function in polar form is given as $r(\phi)$, which denotes the dependence of the radius r on the angle ϕ. For the coordinates x and y we have:

$$y = r(\phi)\ \sin(\phi)$$
$$x = r(\phi)\ \cos(\phi)$$



According to our previous considerations this means that we can consider this as a function in parametric form with the parameter ϕ, and thus:

$$\frac{\partial y}{\partial x} = \frac{\frac{\partial y}{\partial \phi}}{\frac{\partial x}{\partial \phi}} = \frac{\frac{\partial r(\phi)\ \sin(\phi)}{\partial \phi}}{\frac{\partial r(\phi)\ \cos(\phi)}{\partial \phi}} = \frac{\frac{\partial r(\phi)}{\partial \phi}\ \sin(\phi) + r(\phi)\ \cos(\phi)}{\frac{\partial r(\phi)}{\partial \phi}\ \cos(\phi) - r(\phi)\ \sin(\phi)}$$

Let's consider as example the logarithmic spiral $r = a\ e^{k\ \phi}$. Its derivative $\frac{\partial r}{\partial \phi}$ *of the polar coordinates written in terms of polar coordinates* is given by:

$$\frac{\partial r}{\partial \phi} = \frac{\partial a\ e^{k\ \phi}}{\partial \phi} = a\ k\ e^{k\ \phi}$$

Its derivative $\frac{\partial y}{\partial x}$ *of the cartesian coordinates written in terms of polar coordinates* is given by:

$$\frac{\partial y}{\partial x} = \frac{\frac{\partial r}{\partial \phi}\ \sin(\phi) + r(\phi)\ \cos(\phi)}{\frac{\partial r}{\partial \phi}\ \cos(\phi) - r(\phi)\ \sin(\phi)} = \frac{a\ k\ e^{k\ \phi}\ \sin(\phi) + a\ e^{k\ \phi}\ \cos(\phi)}{a\ k\ e^{k\ \phi}\ \cos(\phi) - a\ e^{k\ \phi}\ \sin(\phi)} =$$
$$\frac{k\ \sin(\phi) + \cos(\phi)}{k\ \cos(\phi) - \sin(\phi)}$$

To obtain the derivative $\frac{\partial r}{\partial \phi}$ we only have to differentiate r for ϕ using one of the many possible methods of the HP49G. That means we can enter $r(\phi)$, then ϕ, and then use ∂ or DERIV, and so on. To obtain the derivative $\frac{\partial y}{\partial x}$, we can use one of the programs dF1F2 or dPARMF, after transforming $r(\phi)$ to its parametric form. And this is easily done.

Let's see what we have to do to use dF1F2 for finding $\frac{\partial y}{\partial x}$ of the logarithmic spiral. Enter a $e^{k\ \phi}$ SIN( ), then a $e^{k\ \phi}$ COS( ). For the character , enter a capital O, press ALPHA if you aren't already in alpha mode, then and 9 to change the capital O to , the character that resembles ϕ most in the HP49G character table. Enter (the parameter) and press dF1F2. The result is:

$$-\frac{k\ \sin(\phi)+\cos(\phi)}{\sin(\phi)-k\ \cos(\phi)}$$

We can make a small program that takes a function in its polar form $r(\phi)$, creates the algebraic objects a $e^{k\ \phi}$ SIN( ) and a $e^{k\ \phi}$ COS( ), and then uses dF1F2 to find the derivative $\frac{\partial y}{\partial x}$.

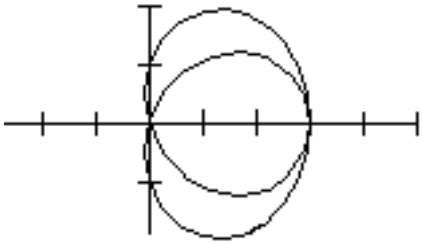
```
<<
  → polarF          @To get  , enter a small o and then
  <<                @press [ALPHA] if not already in
    polarF   SIN *  @alpha more, then [red shift], [9]
    polarF   COS *

    dF1F2
  >>
>>
```

This is the program dYXr . Let's try it with a function in polar form. We use the function r = COS( )^2 + 3 COS( ) − 1. Enter COS( )^2 + 3 COS( ) − 1, then , and press dYXr⊗ to get:

$$\frac{(2\ \mathrm{COS}(\ )+3)\ \mathrm{SIN}(\ )^2-(\mathrm{COS}(\ )^3+3\ \mathrm{COS}(\ )^2-\mathrm{COS}(\ ))}{(3\ \mathrm{COS}(\ )^2+6\ \mathrm{COS}(\ )-1)\ \mathrm{SIN}(\ )}$$

How does r = COS( )^2 + 3 COS( ) − 1 look, anyway? For plotting such polar functions the HP49G has the built-in plot type polar. Let's do a polar plot. Go to the PLOT SETUP screen and select Polar plot type. Enter COS( )^2 + 3 COS( ) − 1 in the input field EQ:. Enter in the input field Indep:. Go to the PLOT WINDOW − POLAR screen and enter horizontal view from −3 to 5 and vertical view from −2 to 2. Press ERASE and then DRAW to plot the polar function. Let's see how the data stored in the system reserved variable PPAR are used when the plot type is polar.



PPAR is a list with 7 items, which the plotting commands use for drawing according to the settings of the user. The list has the form:

{(xmin,ymin) (xmax,ymax) indep res axes ptype depend}

For the plot type POLAR these elements have the following meaning:

(xmin,ymin) A complex number which specifies the lower left corner of the display range. Default value is (−6.5, −3.1). The programmable command for setting this parameter is PMIN. This command takes a complex number from the stack and puts it in the first position of PPAR.

(xmax,ymax) As you might have imagine, a complex number which specifies the upper right corner of the display range. Default value is (6.5,3.2). The programmable command for setting this parameter is PMAX. This command takes a complex number from the stack and puts it in the second position of PPAR.

There are two additional commands for setting up the display range. The first is XRNG. It takes two real numbers from the stack (the minimum and maximum of the view range of the X-axis) and sets up the parameters of and of . The other is YRNG and as you can think it does the same for the Y-axis.

indep

This element can be either a name specifying the independent variable of the expression that we want to plot. Or it can be a list which contains the name of the independent variable and the minimum and maximum of the plotting range. This allows to have different values for the viewing and plotting range. Default for this parameter is X. There are two programmable commands that can be used for setting this parameter. We have the command INDEP, which can take as arguments:

- 1) The name of the independent variable. If a name is given to INDEP then this name replaces the third element of PPAR, that is if you already have specified a plotting range, then this will be lost and the viewing range from the parameters (xmin,ymin) and (xmax,ymax) will be used.
- 2) A list which contains the name of the independent variable. In this case the independent variable is replaced but an existing plotting range will not be touched.
- 3) A list with the name of the independent variable, a real number that specifies the minimum of the plotting range and a real number that specifies the maximum of the plotting range.
- 4) A list with two numbers specifying the minimum and maximum of the plotting range. The independent variable remains untouched.
- 5) Two real numbers that specify the minimum and the maximum of the plotting range. The independent variable remains untouched.

res

A real number that specifies the interval in user coordinates between the values of the independent variable. The default value is 0 and specifies an interval of 2 degrees, 2 grads or $\frac{\pi}{90}$ radians. The command that sets this parameter is RES.

axes

This element is either a complex number specifying the coordinates of intersection of the axes. Or 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. Commands for this parameter are: AXES, which takes as arguments a complex number representing the coordinates of axes intersection, or a list that has the parameters listed above. ATICK, which sets up the distance between tick marks on the axes. This command takes as arguments either a real number that specifies the distance between tick marks in user units for both axes, or a list with two real numbers that specify this distance separately for the X- and Y-axis, or a binary integer that specifies the distance between tick marks in pixels for both axes, or a list with two binary integers that specify this distance separately for the X- and Y-axis.

ptype

One of the plot types available on the HP49G out of

the box. These are: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE and FAST−3D. The commands for setting the appropriate plot type are the same like the parameters above, that is, if you want to set up the plot type polar from a program, you just enter the command POLAR.

depend — A name that specifies the dependent variable. Default for this parameter is Y. The command for setting the dependent variable is DEPND and its arguments have the same forms as the arguments for INDEP. Note that a plot range for the independent variable is only used for the plot type TRUTH but is ignored otherwise.

We examined a lot of things about the capabilities of the HP49G when it comes to derivatives, but we still didn't answer a simple question. The HP49G has a huge amount of built-in functions. Which of them can it differentiate out of the box? As we have seen, there are at least two built-in functions that it doesn't know how to differentiate, namely GAMMA and !. For most of the built-in functions the HP49G provides a derivative. So most of the time you will just use DERIV or DERVX or ∂ in any possible syntax and the HP49G will find the derivative. But for some functions you will have to define the derivative the way we did for GAMMA, because the HP49G doesn't provide a built-in derivative for them. There are also some built-in functions which show an unusual behaviour. Enter X, Y and press %. The HP49G returns:

$$X\ \frac{Y}{100}$$

which of course can be differentiated for X or Y using one of the differentiation commands. For example enter X and press DERIV to get:

$$\frac{Y}{100}$$

But if you enter the algebraic object %(X,Y), then X, and use ∂ or DERIV, then the result is d1%(X,Y), though the HP49G knows how to differentiate the function % because it knows its simple definition:

$$X\ \frac{Y}{100}$$

If you enter the algebraic object:

$\frac{\partial}{\partial X}(\%(X,Y))$ and expand or evaluate, then the result will be again d1%(X,Y). We would expect that expanding or evaluating DERIV(%(X,Y),X) also returns d1%(X,Y), wouldn't we? But no, this time the HP49G returns the result:

$$\frac{Y}{100}$$

And so we have yet another unexpected and puzzling behaviour. Sometimes the one way, some times the other way. Let's say that this machine has "character" ;-) The functions for which the HP49G doesn't provide a derivative, or for which it behaves the above ambiguous way are summarised on the tables from the next page up to page 2-106. Those functions are only functions in the HP49G sense, that means that they are allowed in algebraics and thus we can formally construct a derivative of them. The red cells contain the cases where the derivative can't be found. The green cells contain the cases where the analytic derivative is returned. The first column contains the functions. If a cell that contains the function is red, that means that you have to make a user

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
!	X and then ! returns X!. Enter X and use ∂ to get d1(X).	X and then ! returns X!. Enter X and use DERIV to get d1(X).	Enter X!, then X, and use ∂ to get d1(X).	Enter X!, then X, and use DERIV to get d1(X).	Enter $\frac{\partial}{\partial X}$(X!) and EXPAND or EVAL to get d1(X).	Enter DERIV(X!,X) and EXPAND or EVAL to get d1(X).
%	X, Y, then % returns $X\ \frac{Y}{100}$. Enter X and use ∂ to get $\frac{Y}{100}$.	X, Y, then % returns $X\ \frac{Y}{100}$. Enter X and use DERIV to get $\frac{Y}{100}$.	Enter %(X,Y), then X, and use ∂ to get d1%(X,Y).	Enter %(X,Y), then X, and use DERIV to get d1%(X,Y).	Enter $\frac{\partial}{\partial X}$(%(X,Y)) and EXPAND or EVAL to get d1%(X,Y).	Enter DERIV(%(X,Y),X) and EXPAND or EVAL to get $\frac{Y}{100}$.
%CH	X, Y, then %CH returns $\left(\frac{Y}{X}-1\right)100$. Enter X and use ∂ to get $100\ \frac{-Y}{SQ(X)}$.	X, Y, then %CH returns $\left(\frac{Y}{X}-1\right)100$. Enter X and use DERIV to get $100\ \frac{-Y}{SQ(X)}$.	Enter %CH(X,Y), then X, and use ∂ to get d1%CH(X,Y).	Enter %CH(X,Y), then X, and use DERIV to get d1%CH(X,Y).	Enter $\frac{\partial}{\partial X}$(%CH(X,Y)) and EXPAND or EVAL to get d1%CH(X,Y).	Enter DERIV(%CH(X,Y),X) and EVAL to get $100\ \frac{-Y}{SQ(X)}$, or EXPAND to get $-\frac{100\ Y}{X^2}$.
%T	X, Y, then %T returns $\frac{Y}{X}\ 100$. Enter X and use ∂ to get $100\ \frac{-Y}{SQ(X)}$.	X, Y, then %T returns $\frac{Y}{X}\ 100$. Enter X and use DERIV to get $100\ \frac{-Y}{SQ(X)}$.	Enter %T(X,Y), then X, and use ∂ to get d1%T(X,Y).	Enter %T(X,Y), then X, and use DERIV to get d1%T(X,Y).	Enter $\frac{\partial}{\partial X}$(%T(X,Y)) and EXPAND or EVAL to get d1%T(X,Y).	Enter DERIV(%T(X,Y),X) and EVAL to get $100\ \frac{-Y}{SQ(X)}$, or EXPAND to get $-\frac{100\ Y}{X^2}$.

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
<	X, Y, and then < returns X < Y. Enter X and use ∂ to get d1<(X,Y).	X, Y, and then < returns X < Y. Enter X and use DERIV to get d1<(X,Y).	Enter X < Y, then X, and use ∂ to get d1<(X,Y).	Enter X < Y, then X, and use DERIV to get d1<(X,Y).	Enter $\frac{\partial}{\partial X}$(X < Y) and EXPAND or EVAL to get d1<(X,Y).	Enter DERIV(X < Y,X) and EXPAND or EVAL to get d1<(X,Y).
==	X, Y, and then == returns X == Y. Enter X and use ∂ to get d1==(X,Y).	X, Y, and then == returns X == Y. Enter X and use DERIV to get d1==(X,Y).	Enter X == Y, then X, and use ∂ to get d1==(X,Y).	Enter X == Y, then X, and use DERIV to get d1==(X,Y).	Enter $\frac{\partial}{\partial X}$(X == Y) and EXPAND or EVAL to get d1==(X,Y).	Enter DERIV(X == Y,X) and EXPAND or EVAL to get d1==(X,Y).
>	X, Y, and then > returns X > Y. Enter X and use ∂ to get d1>(X,Y).	X, Y, and then > returns X > Y. Enter X and use DERIV to get d1>(X,Y).	Enter X > Y, then X, and use ∂ to get d1>(X,Y).	Enter X > Y, then X, and use DERIV to get d1>(X,Y).	Enter $\frac{\partial}{\partial X}$(X > Y) and EXPAND or EVAL to get d1>(X,Y).	Enter DERIV(X > Y,X) and EVAL or EXPAND to get d1>(X,Y).
AND	X, Y, and then AND returns X AND Y. Enter X and use ∂ to get d1AND(X,Y).	X, Y, and then AND returns X AND Y. Enter X and use DERIV to get d1AND(X,Y).	Enter X AND Y, then X, and use ∂ to get d1AND(X,Y).	Enter X AND Y, then X, and use DERIV to get d1AND(X,Y).	Enter $\frac{\partial}{\partial X}$(X AND Y) and EXPAND or EVAL to get d1AND(X,Y).	Enter DERIV(X AND Y,X) and EXPAND or EVAL to get d1AND(X,Y).
APPLY	{X Y}, Y, then APPLY returns F(X,Y). Enter X and use ∂ to get d1F(X,Y).	{X Y}, Y, then APPLY returns F(X,Y). Enter X and use DERIV to get d1F(X,Y).	Enter APPLY(F,X,Y) from the command line to get Invalid Expression. Enter X, and use ∂ to get d1F(X,Y).	Enter APPLY(F,X,Y) from the command line to get Invalid Expression. Enter X, and use DERIV to get d1F(X,Y).	Enter ∂X(APPLY(F,X,Y)) from the command line to get ∂X(Invalid Expression). EXPAND or EVAL to get d1APPLY(F,{X Y}).	Enter DERIV(APPLY(F,X,Y),X) from the command line to get Invalid Expression. EXPAND or EVAL to get d1F(X,Y).

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
CEIL	X then CEIL returns CEIL(X). Enter X and use ∂ to get d1CEIL(X).	X then CEIL returns CEIL(X). Enter X and use DERIV to get d1CEIL(X).	Enter CEIL(X), then X, and use ∂ to get d1CEIL(X).	Enter CEIL(X), then X, and use DERIV to get d1CEIL(X).	Enter $\frac{\partial}{\partial X}$(CEIL(X)) and EXPAND or EVAL to get d1CEIL(X).	Enter DERIV(CEIL(X),X) and EXPAND or EVAL to get d1CEIL(X).
COMB	X, Y, and then COMB returns $\frac{X!}{(X-Y)!\,Y!}$. Enter X and use ∂ to get $\frac{\left(\frac{X!\,d1(X)}{(X-Y)!}-\frac{X!\,d1(X-Y)}{(X-Y)!^2}\right)Y!}{SQ((X-Y)!\,Y!)}$.	X, Y, and then COMB returns $\frac{X!}{(X-Y)!\,Y!}$. Enter X and use DERIV to get $\frac{\left(\frac{X!\,d1(X)}{(X-Y)!}-\frac{X!\,d1(X-Y)}{(X-Y)!^2}\right)Y!}{SQ((X-Y)!\,Y!)}$.	Enter COMB(X,Y), then X, and use ∂ to get d1COMB(X,Y).	Enter COMB(X,Y), then X, and use DERIV to get d1COMB(X,Y).	Enter $\frac{\partial}{\partial X}$(COMB(X,Y)) and EXPAND or EVAL to get d1COMB(X,Y).	Enter DERIV(COMB(X,Y),X) and EXPAND to get $\left(\frac{X!\,d1(X-Y)}{(X-Y)!}-\frac{X!\,d1(X)}{Y!\,(X-Y)!^2}\right)Y!$ or EVAL to get $\frac{\left(\frac{X!\,d1(X)}{(X-Y)!}-\frac{X!\,d1(X-Y)}{(X-Y)!^2}\right)Y!}{SQ((X-Y)!\,Y!)}$.
CONJ	In real mode enter X+Y i, then CONJ to get X+-Y i. Enter X and use ∂ to get 1. In complex mode enter Z, then CONJ to get CONJ(Z). Enter Z and use ∂ to get d1CONJ(Z).	In real mode enter X+Y i, then CONJ to get X+-Y i. Enter X and use DERIV to get 1. In complex mode enter Z, then CONJ to get CONJ(Z). Enter Z and use DERIV to get d1CONJ(Z).	In real mode enter CONJ(X+Y i), then X and use ∂ to get d1CONJ(X+i Y). In complex mode enter CONJ(Z), and use ∂ to get d1CONJ(Z).	In real mode enter CONJ(X+Y i), then X, and use DERIV to get d1CONJ(X+i Y). In complex mode enter CONJ(Z), and use ∂ to get d1CONJ(Z).	In real mode enter $\frac{\partial}{\partial X}$(CONJ(X+Y i)), and use EXPAND or EVAL to get d1CONJ(X+i Y). In complex mode enter $\frac{\partial}{\partial X}$(CONJ(Z)), and use EVAL or EXPAND to get d1CONJ(Z).	In real mode enter DERIV(CONJ(X+Y i),X), and use EXPAND or EVAL to get 1. In complex mode enter DERIV(CONJ(Z),Z), and use EVAL or EXPAND to get d1CONJ(Z).

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
CYCLOTOMIC	3 then CYCLOTOMIC returns X^2+X+1. Enter X and use ∂ to get $2X+1$.	3 then CYCLOTOMIC returns X^2+X+1. Enter X and use DERIV to get $2X+1$.	Enter CYCLOTOMIC(3), then X, and use ∂ to get $2X+1$.	Enter CYCLOTOMIC(3), then X, and use DERIV to get $2X+1$.	Enter $\frac{\partial}{\partial X}$(CYCLOTOMIC(3)) and EXPAND or EVAL to get 0.	Enter DERIV(CYCLOTOMIC(3),X) and EXPAND or EVAL to get $2X+1$.
DARCY	X, Y, and then DARCY returns DARCY(X,Y). Enter X and use ∂ to get d1DARCY(X,Y).	X, Y, and then DARCY returns DARCY(X,Y). Enter X and use DERIV to get d1DARCY(X,Y).	Enter DARCY(X,Y), then X, and use ∂ to get d1DARCY(X,Y).	Enter DARCY(X,Y), then X, and use DERIV to get d1DARCY(X,Y).	Enter $\frac{\partial}{\partial X}$(DARCY(X,Y)) and EXPAND or EVAL to get d1DARCY(X,Y).	Enter DERIV(DARCY(X,Y),X) and EXPAND or EVAL to get d1DARCY(X,Y).
DEF	Enter $F(X)=X^2-X$ and then DEF. Enter F(X), then X and use ∂ to get $2X-1$.	Enter $F(X)=X^2-X$ and then DEF. Enter F(X), then X and use DERIV to get $2X-1$.	From the EQW enter DEF(F(X)=X^2-X) and then enter X and use ∂ to get $2X-1=2X-1$. This also creates the user defined function.	From the EQW enter DEF(F(X)=X^2-X) and then enter X and use DERIV to get $2X-1=2X-1$. This also creates the user defined function.	From the EQW enter $\frac{\partial}{\partial X}$(DEF(F(X)=X^2-X)) and then EVAL to get $2X-1=2X-1$. If you use EXPAND, you get 0. Both EVAL and EXPAND also create the user defined function.	From the EQW enter DERIV(DEF(F(X)=X^2-X),X) and then EVAL or EXPAND to get $2X-1=2X-1$. This also creates the user defined function.
DROITE	DROITE works only with arguments evaluable to reals or complex.	DROITE works only with arguments evaluable to reals or complex.	Enter DROITE(X,Y), then X, and use ∂ to get d1DROITE(X,Y).	Enter DROITE(X,Y), then X, and use DERIV to get d1DROITE(X,Y).	Enter $\frac{\partial}{\partial X}$(DROITE(X,Y)) and EXPAND or EVAL to get d1DROITE(X,Y).	Enter DERIV(DROITE(X,Y), and EXPAND or EVAL. The HP49G errors out with "Bad Argument Type"
D → R	X, and then D → R returns D → R(X). Enter X and use ∂ to get d1D → R(X).	X, and then D → R returns D → R(X). Enter X and use DERIV to get d1D → R(X).	Enter D → R(X), then X, and use ∂ to get d1D → R(X).	Enter D → R(X), then X, and use DERIV to get d1D → R(X).	Enter $\frac{\partial}{\partial X}$(D → R(X)) and EXPAND or EVAL to get d1D → R(X).	Enter DERIV(D → R(X),X) and EXPAND or EVAL to get d1D → R(X).

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
EULER	EULER works only with integer arguments.	EULER works only with integer arguments.	Enter EULER(X), then X, and use ∂ to get d1EULER(X).	Enter EULER(X), then X, and use DERIV to get d1EULER(X).	Enter $\frac{\partial}{\partial X}$(EULER(X)) and then EVAL or EXPAND to get d1EULER(X).	Enter DERIV(EULER(X),X) and then EVAL or EXPAND. The HP49G errors out "Bad Argument Type"
F0λ	X, Y then F0λ returns F0λ(X,Y). Enter X and use ∂ to get d1F0λ(X,Y).	X, Y then F0λ returns F0λ(X,Y). Enter X and use DERIV to get d1F0λ(X,Y).	Enter F0λ(X,Y), then X, and use ∂ to get d1F0λ(X,Y).	Enter F0λ(X,Y), then X, and use DERIV to get d1F0λ(X,Y).	Enter $\frac{\partial}{\partial X}$(F0λ(X,Y)) and then EVAL or EXPAND to get d1F0λ(X,Y).	Enter DERIV(F0λ(X,Y),X) and then EVAL or EXPAND to get d1F0λ(X,Y).
FACT	X then FACT returns X!. Enter X and use ∂ to get d1(X).	X then FACT returns X!. Enter X and use DERIV to get d1(X).	Enter FACT(X), then X, and use ∂ to get d1FACT(X).	Enter FACT(X), then X, and use DERIV to get d1FACT(X).	Enter $\frac{\partial}{\partial X}$(FACT(X)) and then EVAL to get d1FACT(X,Y). EXPAND returns d1(X).	Enter DERIV(FACT(X),X) and then EVAL or EXPAND to get d1(X).
FANNING	X, Y then FANNING returns FANNING(X,Y). Enter X and use ∂ to get d1FANNING(X,Y).	X, Y then FANNING returns FANNING(X,Y). Enter X and use DERIV to get d1FANNING(X,Y).	Enter FANNING(X,Y), then X, and use ∂ to get d1FANNING(X,Y).	Enter FANNING(X,Y), then X, and use DERIV to get d1FANNING(X,Y).	Enter $\frac{\partial}{\partial X}$(FANNING(X,Y)) and then EVAL or EXPAND to get d1FANNING(X,Y).	Enter DERIV(FANNING(X,Y),X) and then EVAL or EXPAND to get d1FANNING(X,Y).
FLOOR	X then FLOOR returns FLOOR(X). Enter X and use ∂ to get d1FLOOR(X).	X then FLOOR returns FLOOR(X). Enter X and use DERIV to get d1FLOOR(X).	Enter FLOOR(X), then X, and use ∂ to get d1FLOOR(X).	Enter FLOOR(X), then X, and use DERIV to get d1FLOOR(X).	Enter $\frac{\partial}{\partial X}$(FLOOR(X)) and EXPAND or EVAL to get d1FLOOR(X).	Enter DERIV(FLOOR(X),X) and EXPAND or EVAL to get d1FLOOR(X).

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
FP	X, then FP returns FP(X). Enter X and use ∂ to get d1FP(X).	X, then FP returns FP(X). Enter X and use DERIV to get d1FP(X).	Enter FP(X), then X, and use ∂ to get d1FP(X).	Enter FP(X), then X, and use DERIV to get d1FP(X).	Enter $\frac{\partial}{\partial X}$(FP(X)) and EXPAND or EVAL to get d1FP(X).	Enter DERIV(FP(X),X) and EXPAND or EVAL to get d1FP(X).
GAMMA	X, and then GAMMA returns GAMMA(X). Enter X and use ∂ to get d1GAMMA(X).	X, and then GAMMA returns GAMMA(X). Enter X and use DERIV to get d1GAMMA(X).	Enter GAMMA(X), then X, and use ∂ to get d1GAMMA(X).	Enter GAMMA(X), then X, and use DERIV to get d1GAMMA(X).	Enter $\frac{\partial}{\partial X}$(GAMMA(X)) and EXPAND or EVAL to get d1GAMMA(X).	Enter DERIV(GAMMA(X),X) and EXPAND or EVAL to get d1GAMMA(X).
GCD	X^2-1, X+1, then GCD returns {X+1}. Enter X and use ∂ to get {}.	X^2-1, X+1, then GCD returns {X+1}. Enter X and use DERIV to get {}.	Enter GCD(X^2-1,X+1), then X, and use ∂ to get 1.	Enter GCD(X^2-1,X+1), then X, and use DERIV to get 1.	Enter $\frac{\partial}{\partial X}$(GCD($X^2-1$,X+1)) and then EVAL to get d2GCD(X^2-1,X+1)+2 X d1GCD(X^2-1,X+1). EXPAND returns 2 X d1GCD(X^2-1,X+1)+d2GCD(X^2-1,X+1).	Enter DERIV(GCD(X^2-1,X and then EVAL or EXPAND to get 1.
HERMITE	3 then HERMITE returns 8 X^3−12 X. Enter X and use ∂ to get 8 3 X^2−12.	3 then HERMITE returns 8 X^3−12 X. Enter X and use DERIV to get 8 3 X^2−12.	Enter HERMITE(3), then X and use ∂ to get 8 3 X^2−12.	Enter HERMITE(3), then X and use DERIV to get 8 3 X^2−12.	Enter $\frac{\partial}{\partial X}$(HERMITE(3)), and use EVAL or EXPAND to get 0.	Enter DERIV(HERMITE(3),X), and use EVAL to get 8 3 X^2−12. EXPAND returns 24 X^2−12.

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
IBERNOULLI	IBERNOULLI works only with integer arguments.	IBERNOULLI works only with integer arguments.	Enter IBERNOULLI(X), then X, and use ∂ to get d1IBERNOULLI(X).	Enter IBERNOULLI(X), then X, and use DERIV to get d1IBERNOULLI(X).	Enter $\frac{\partial}{\partial X}$(IBERNOULLI(X)) and EXPAND or EVAL to get d1IBERNOULLI(X).	Enter DERIV(IBERNOULLI(X),X) and EXPAND or EVAL. The HP49G errors out "Bad Argument Type"
IFTE	A, X^2, X^3 and then IFTE returns IFTE(A,X^2,X^3). Enter X and use ∂ to get IFTE(A,2 X,3 X^2).	A, X^2, X^3 and then IFTE returns IFTE(A,X^2,X^3). Enter X and use DERIV to get IFTE(A,2 X,3 X^2).	Enter IFTE(A,X^2,X^3), then X, and use ∂ to get IFTE(A,2 X,3 X^2).	Enter IFTE(A,X^2,X^3), then X, and use DERIV to get IFTE(A,2 X,3 X^2).	Enter $\frac{\partial}{\partial X}$(IFTE(A,$X^2$,$X^3$)) and EXPAND to get IFTE(A,2 X,3 X^2) EVAL errors out with "CAS Internal Error".	Enter DERIV(IFTE(A,X^2,X^3),X) and EXPAND or EVAL to get IFTE(A,2 X,3 X^2).
IM	In real mode enter $X+X^2$ i, then IM to get X^2. Enter X and use ∂ to get 2 X. In complex mode enter Z, then IM to get IM(Z). Enter Z and use ∂ to get d1IM(Z).	In real mode enter $X+X^2$ i, then IM to get X^2. Enter X and use DERIV to get 2 X. In complex mode enter Z, then IM to get IM(Z). Enter Z and use DERIV to get d1IM(Z).	In real mode enter IM($X+X^2$ i), then X and use ∂ to get d1IM($X+X^2$ i). In complex mode enter IM(Z), then Z and use ∂ to get d1IM(Z).	In real mode enter IM($X+X^2$ i), then X and use DERIV to get d1IM($X+X^2$ i). In complex mode enter IM(Z), then Z and use ∂ to get d1IM(Z).	In real mode enter $\frac{\partial}{\partial X}$(IM($X+X^2$ i)) and then EVAL or EXPAND to get d1IM($X+X^2$ i). In complex mode enter $\frac{\partial}{\partial Z}$(IM(Z)) and then EVAL or EXPAND to get d1IM(Z).	In real mode enter DERIV(IM($X+X^2$ i),X) and then EVAL or EXPAND to get 2 X. In complex mode enter DERIV(IM(Z),Z) and then EVAL or EXPAND to get d1IM(Z).

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
INT	X^2, X, X, and then INT returns $\frac{1}{3} X^3$. Enter X and use ∂ to get $\frac{1}{3} 3 X^2$.	X^2, X, X, and then INT returns $\frac{1}{3} X^3$. Enter X and use DERIV to get $\frac{1}{3} 3 X^2$.	Enter INT(X^2,X,X), then X, and use ∂. The HP49G errors out "Can't derive int. var"	Enter INT(X^2,X,X), then X, and use DERIV. The HP49G errors out "Can't derive int. var"	Enter $\frac{\partial}{\partial X}$(INT($X^2$,X,X)) and EXPAND to get $\frac{3 3 X^2}{9}$. EVAL errors out with "Can't derive int. var"	Enter DERIV(INT(X^2,X,X),X) and EVAL to get $\frac{3 3 X^2}{9}$. EXPAND returns X^2.
IP	X and then IP returns IP(X). Enter X and use ∂ to get d1IP(X).	X and then IP returns IP(X). Enter X and use DERIV to get d1IP(X).	Enter IP(X), then X, and use ∂ to get d1IP(X).	Enter IP(X) then X, and use DERIV to get d1IP(X).	Enter $\frac{\partial}{\partial X}$(IP(X)) and EVAL or EXPAND to get d1IP(X).	Enter DERIV(IP(X),X) and EVAL or EXPAND to get d1IP(X).
IQUOT	IQUOT works only with integer arguments.	IQUOT works only with integer arguments.	Enter IQUOT(X,Y), then X, and use ∂ to get d1IQUOT(X,Y).	Enter IQUOT(X,Y), then X, and use DERIV to get d1IQUOT(X,Y).	Enter $\frac{\partial}{\partial X}$(IQUOT(X,Y)) and EVAL or EXPAND to get d1IQUOT(X,Y).	Enter DERIV(IQUOT(X,Y),X) and EVAL or EXPAND. The HP49G errors out with "Bad Argument Type".
IREMAINDER	IREMAINDER works only with integer arguments.	IREMAINDER works only with integer arguments.	Enter IREMAINDER(X,Y), then X, and use ∂ to get d1IREMAINDER(X,Y).	Enter IREMAINDER(X,Y), then X, and use DERIV to get d1IREMAINDER(X,Y).	Enter $\frac{\partial}{\partial X}$(IREMAINDER(X,Y)) and EVAL or EXPAND to get d1IREMAINDER(X,Y).	Enter DERIV(IREMAINDER(X,Y),X) and EVAL or EXPAND. The HP49G errors out with "Bad Argument Type".
ISPRIME?	ISPRIME? works only with integer arguments.	ISPRIME? works only with integer arguments.	Enter ISPRIME?(X), then X, and use ∂ to get d1ISPRIME?(X).	Enter ISPRIME?(X), then X, and use DERIV to get d1ISPRIME?(X).	Enter $\frac{\partial}{\partial X}$(ISPRIME?(X)), then X, EVAL or EXPAND to get d1ISPRIME?(X).	Enter DERIV(ISPRIME?(X),X), then X, EVAL or EXPAND. The HP49G errors out with "Bad Argument Type".

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
I → R	I → R works only with integer arguments.	I → R works only with integer arguments.	Enter I → R(X), then X, and use ∂ to get d1 → R(X).	Enter I → R(X), then X, and use DERIV to get d1 → R(X).	Enter $\frac{\partial}{\partial X}$(I → R(X)) and EVAL or EXPAND to get d1 → R(X).	Enter DERIV(I → R(X),X) and EVAL or EXPAND to get d1 → R(X).
LEGENDRE	LEGENDRE works only with integer arguments.	LEGENDRE works only with integer arguments.	Enter LEGENDRE(X), then X, and use ∂ to get d1LEGENDRE(X).	Enter LEGENDRE(X), then X, and use DERIV to get d1LEGENDRE(X).	Enter $\frac{\partial}{\partial X}$(LEGENDRE(X)) and EVAL or EXPAND to get d1LEGENDRE(X).	Enter DERIV(LEGENDRE(X),X) and EVAL or EXPAND. The HP49G errors out with "Bad Argument Type".
MANT	X, and then MANT returns MANT(X). Enter X and use ∂ to get d1MANT(X).	X, and then MANT returns MANT(X). Enter X and use DERIV to get d1MANT(X).	Enter MANT(X), then X, and use ∂ to get d1MANT(X).	Enter MANT(X), then X, and use DERIV to get d1MANT(X).	Enter $\frac{\partial}{\partial X}$(MANT(X)) and EVAL or EXPAND to get d1MANT(X).	Enter DERIV(MANT(X),X) and EVAL or EXPAND to get d1QUOT(X,Y).
MAX	X, Y and then MAX returns MAX(X,Y). Enter X and use ∂ to get d1MAX(X,Y).	X, Y and then MAX returns MAX(X,Y). Enter X and use DERIV to get d1MAX(X,Y).	Enter MAX(X,Y), then X, and use ∂ to get d1MAX(X,Y).	Enter MAX(X,Y), then X, and use DERIV to get d1MAX(X,Y).	Enter $\frac{\partial}{\partial X}$(MAX(X,Y)) and EVAL or EXPAND to get d1MAX(X,Y).	Enter DERIV(MAX(X,Y),X) and EVAL or EXPAND to get d1MAX(X,Y).
MIN	X, Y and then MIN returns MIN(X,Y). Enter X and use ∂ to get d1MIN(X,Y).	X, Y and then MIN returns MIN(X,Y). Enter X and use DERIV to get d1MIN(X,Y).	Enter MIN(X,Y), then X, and use ∂ to get d1MIN(X,Y).	Enter MIN(X,Y), then X, and use DERIV to get d1MIN(X,Y).	Enter $\frac{\partial}{\partial X}$(MIN(X,Y)) and EVAL or EXPAND to get d1MIN(X,Y).	Enter DERIV(MIN(X,Y),X) and EVAL or EXPAND to get d1MIN(X,Y).

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
MOD	X, Y and then MOD returns X MOD Y. Enter X and use ∂ to get d1MOD(X,Y).	X, Y and then MOD returns X MOD Y. Enter X and use DERIV to get d1MOD(X,Y).	Enter X MOD Y, then X, and use ∂ to get d1MOD(X,Y).	Enter X MOD Y, then X, and use DERIV to get d1MOD(X,Y).	Enter $\frac{\partial}{\partial X}$(X MOD Y) and EVAL or EXPAND to get d1MOD(X,Y).	Enter DERIV(X MOD Y,X) and EVAL or EXPAND to get d1MOD(X,Y).
NEXTPRIME	NEXTPRIME works only with integer arguments.	NEXTPRIME works only with integer arguments.	Enter NEXTPRIME(X), then X, and use ∂ to get d1NEXTPRIME(X).	Enter NEXTPRIME(X), then X, and use DERIV to get d1NEXTPRIME(X).	Enter $\frac{\partial}{\partial X}$(NEXTPRIME(X)) and EVAL or EXPAND to get d1NEXTPRIME(X).	Enter DERIV(NEXTPRIME(X),X) and EVAL or EXPAND. The HP49G errors out with "Bad Argument Type".
NOT	X, and then NOT returns NOT X. Enter X and use ∂ to get d1NOT(X).	X, and then NOT returns NOT X. Enter X and use DERIV to get d1NOT(X).	Enter NOT X, then X, and use ∂ to get d1NOT(X).	Enter NOT X, then X, and use DERIV to get d1NOT(X).	Enter $\frac{\partial}{\partial X}$(NOT X) and EVAL or EXPAND to get d1NOT(X).	Enter DERIV(NOT X,X) and EVAL or EXPAND to get d1NOT(X).
OR	X, Y, and then OR returns X OR Y. Enter X and use ∂ to get d1OR(X,Y).	X, Y, and then OR returns X OR Y. Enter X and use DERIV to get d1OR(X,Y).	Enter X OR Y, then X, and use ∂ to get d1OR(X,Y).	Enter X OR Y, then X, and use DERIV to get d1OR(X,Y).	Enter $\frac{\partial}{\partial X}$(X OR Y) and EVAL or EXPAND to get d1OR(X,Y).	Enter DERIV(X OR Y,X) and EVAL or EXPAND to get d1OR(X,Y).
PA2B2	PA2B2 works only with integer arguments.	PA2B2 works only with integer arguments.	Enter PA2B2(X), then X, and use ∂ to get d1PA2B2(X).	Enter PA2B2(X), then X, and use DERIV to get d1PA2B2(X).	Enter $\frac{\partial}{\partial X}$(PA2B2(X)) and EVAL or EXPAND to get d1PA2B2(X).	Enter DERIV(PA2B2(X),X) and EVAL or EXPAND. The HP49G errors out with "Bad Argument Type".

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
PERM	X, Y and then PERM returns $\frac{X!}{(X-Y)!}$. Enter X and use ∂ to get $\frac{(X-Y)!\,d1(X)-X!\,d1(X-Y)}{SQ((X-Y)!)}$.	X, Y and then PERM returns $\frac{X!}{(X-Y)!}$. Enter X and use DERIV to get $\frac{(X-Y)!\,d1(X)-X!\,d1(X-Y)}{SQ((X-Y)!)}$.	Enter PERM(X,Y), then X, and use ∂ to get d1PERM(X,Y).	Enter PERM(X,Y), then X, and use DERIV to get d1PERM(X,Y).	Enter $\frac{\partial}{\partial X}$(PERM(X,Y)) and EVAL or EXPAND to get d1PERM(X,Y).	Enter DERIV(PERM(X,Y),X) and EVAL to get $\frac{(X-Y)!\,d1(X)-X!\,d1(X-Y)}{SQ((X-Y)!)}$. EXPAND returns $\frac{X!\,d1(X-Y)}{(X-Y)!}-\frac{(X-Y)!\,d1(X)}{(X-Y)!^2}$.
PREVPRIME	PREVPRIME works only with integer arguments.	NEXTPRIME works only with integer arguments.	Enter PREVPRIME(X), then X, and use ∂ to get d1PREVPRIME(X).	Enter PREVPRIME(X), then X, and use DERIV to get d1PREVPRIME(X).	Enter $\frac{\partial}{\partial X}$(PREVPRIME(X)) and EVAL or EXPAND to get d1PREVPRIME(X).	Enter DERIV(PREVPRIME(X),X) and EVAL or EXPAND. The HP49G errors out with "Bad Argument Type".
PSI	X, 2, and then PSI returns PSI(X,2). Enter X and use ∂ to get d1PSI(X,2).	X, 2, and then PSI returns PSI(X,2). Enter X and use DERIV to get d1PSI(X,2).	Enter PSI(X,2), then X, and use ∂ to get d1PSI(X,2).	Enter PSI(X,2), then X, and use DERIV to get d1PSI(X,2).	Enter $\frac{\partial}{\partial X}$(PSI(X,2)) and EVAL or EXPAND to get d1PSI(X,2).	Enter DERIV(PSI(X,2),X) and EVAL or EXPAND to get d1PSI(X,2).

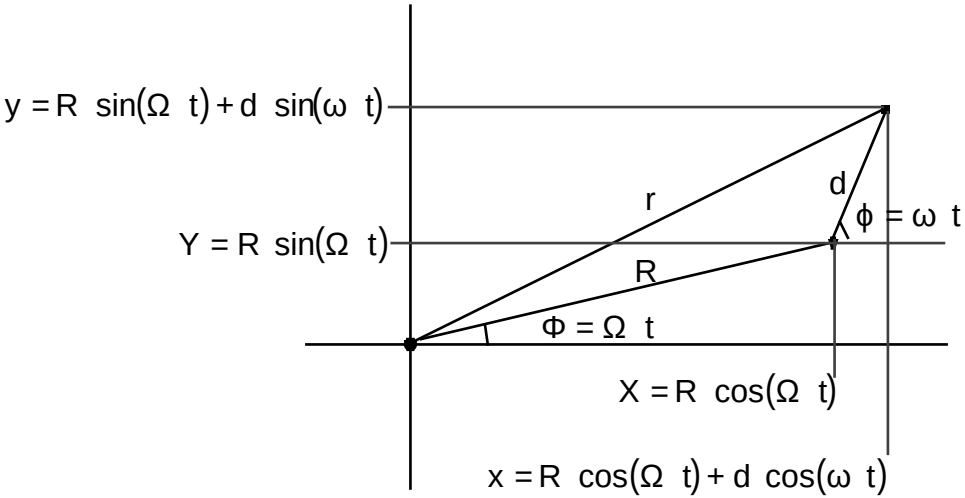
Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
Psi	X, and then Psi returns Psi(X). Enter X and use ∂ to get d1Psi(X).	X, and then Psi returns Psi(X). Enter X and use DERIV to get d1Psi(X).	Enter Psi(X), then X, and use ∂ to get d1Psi(X).	Enter Psi(X), then X, and use DERIV to get d1Psi(X).	Enter $\frac{\partial}{\partial X}$(Psi(X)) and EVAL or EXPAND to get d1Psi(X).	Enter DERIV(Psi(X),X) and EVAL or EXPAND to get d1Psi(X).
RE	In real mode enter $X+X^2\ i$, then RE to get X. Enter X and use ∂ to get 1. In complex mode enter Z, then RE to get RE(Z). Enter Z and use ∂ to get d1RE(Z).	In real mode enter $X+X^2\ i$, then RE to get X. Enter X and use DERIV to get 1. In complex mode enter Z, then RE to get RE(Z). Enter Z and use DERIV to get d1RE(Z).	In real mode enter RE($X+X^2\ i$), then X and use ∂ to get d1RE($X+X^2\ i$). In complex mode enter RE(Z), then Z and use ∂ to get d1RE(Z).	In real mode enter RE($X+X^2\ i$), then X and use DERIV to get d1RE($X+X^2\ i$). In complex mode enter RE(Z), then Z and use DERIV to get d1RE(Z).	In real mode enter $\frac{\partial}{\partial X}$(RE($X+X^2\ i$)) and then EVAL or EXPAND to get d1RE($X+X^2\ i$). In complex mode enter $\frac{\partial}{\partial Z}$(RE(Z)) and then EVAL or EXPAND to get d1RE(Z).	In real mode enter DERIV(RE($X+X^2\ i$),X) and then EVAL or EXPAND to get 1. In complex mode enter $\frac{\partial}{\partial Z}$(RE(Z)) and then EVAL or EXPAND to get d1RE(Z).
RND	X, 2 and then RND returns RND(X,2.). Enter X and use ∂ to get d1RND(X,2.).	X, 2 and then RND returns RND(X,2.). Enter X and use DERIV to get d1RND(X,2.).	Enter RND(X,2), then X, and use ∂ to get d1RND(X,2).	Enter RND(X,2), then X, and use DERIV to get d1RND(X,2).	Enter $\frac{\partial}{\partial X}$(RND(X,2)) and EVAL or EXPAND to get d1RND(X,2).	Enter DERIV(RND(X,2),X) and EVAL or EXPAND to get d1RND(X,2.).
R → D	X, and then R → D returns R → D(X). Enter X and use ∂ to get d1R → D(X).	X, and then R → D returns R → D(X). Enter X and use DERIV to get d1R → D(X).	Enter R → D(X), then X, and use ∂ to get d1R → D(X).	Enter R → D(X), then X, and use DERIV to get d1R → D(X).	Enter $\frac{\partial}{\partial X}$(R → D(X)) and EXPAND or EVAL to get d1R → D(X).	Enter DERIV(R → D(X),X) and EXPAND or EVAL to get d1R → D(X).

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	∂/∂X (Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
R → I	R → I works only with integer arguments.	R → I works only with integer arguments.	Enter R → I(X), then X, and use ∂ to get d1R → I(X).	Enter R → I(X), then X, and use DERIV to get d1R → I(X).	Enter ∂/∂X (R → I(X)) and EVAL or EXPAND to get d1R → I(X).	Enter DERIV(R → I(X),X) and EVAL or EXPAND to get d1R → I(X).
SIDENS	X, and then SIDENS returns SIDENS(X). Enter X and use ∂ to get d1SIDENS(X).	X, and then SIDENS returns SIDENS(X). Enter X and use DERIV to get d1SIDENS(X).	Enter SIDENS(X), then X, and use ∂ to get d1SIDENS(X).	Enter SIDENS(X), then X, and use DERIV to get d1SIDENS(X).	Enter ∂/∂X (SIDENS(X)) and EVAL or EXPAND to get d1SIDENS(X).	Enter DERIV(SIDENS(X),X) and EVAL or EXPAND to get d1SIDENS(X).
TCHEBYCHEFF	2 and then TCHEBYCHEFF returns $2\ X^2-1$. Enter X and use ∂ to get 2 2 X.	2 and then TCHEBYCHEFF returns $2\ X^2-1$. Enter X and use DERIV to get 2 2 X.	Enter TCHEBYCHEFF(2), then X, and use ∂ to get 2 2 X.	Enter TCHEBYCHEFF(2), then X, and use DERIV to get 2 2 X.	Enter ∂/∂X (TCHEBYCHEFF(2)) and EVAL or EXPAND to get 0.	Enter DERIV(TCHEBYCHEFF(2),X) and EVAL to get 2 2 X. EXPAND returns 4 X.
TDELTA	X, Y, and then TDELTA returns TDELTA(X,Y). Enter X and use ∂ to get d1TDELTA(X,Y).	X, Y, and then TDELTA returns TDELTA(X,Y). Enter X and use DEIV to get d1TDELTA(X,Y).	Enter TDELTA(X,Y), then X, and use ∂ to get d1TDELTA(X,Y).	Enter TDELTA(X,Y), then X, and use DERIV to get d1TDELTA(X,Y).	Enter ∂/∂X (TDELTA(X,Y)) and EVAL or EXPAND to get d1TDELTA(X,Y).	Enter DERIV(TDELTA(X,Y),X) and EVAL or EXPAND to get d1TDELTA(X,Y).
TRNC	X, 2, and then TRNC returns TRNC(X,2.). Enter X and use ∂ to get d1TRNC(X,2.).	X, 2, and then TRNC returns TRNC(X,2.). Enter X and use DERIV to get d1TRNC(X,2.).	Enter TRNC(X,2), then X, and use ∂ to get d1TRNC(X,2).	Enter TRNC(X,2), then X, and use DERIV to get d1TRNC(X,2).	Enter ∂/∂X (TRNC(X,2)) and EVAL or EXPAND to get d1TRNC(X,2).	Enter DERIV(TRNC(X,2),X) and EVAL or EXPAND to get d1TRNC(X,2.).

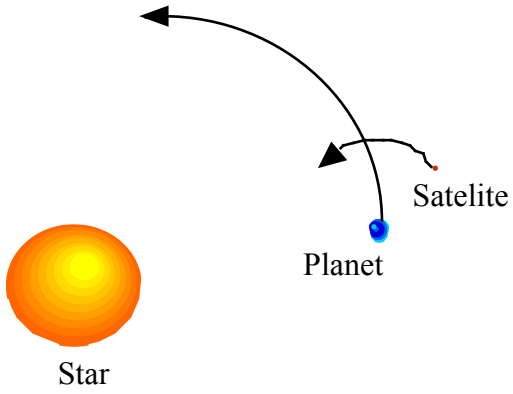
Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	∂/∂X (Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
UBASE	In the command line enter $X^2\ 1\_cm^2$ and use UBASE to get UBASE(X^2 1_cm²). Enter X and use ∂ to get d1UBASE(X^2 1_cm²).	In the command line enter $X^2\ 1\_cm^2$ and use UBASE to get UBASE(X^2 1_cm²). Enter X and use DERIV to get d1UBASE(X^2 1_cm²).	In the command line enter UBASE(X^2 1_cm²), then X, and use ∂ to get d1UBASE(X^2 1_cm²).	In the command line enter UBASE(X^2 1_cm²), then X, and use DERIV to get d1UBASE(X^2 1_cm²).	In the command line enter $\frac{\partial}{\partial X}$(UBASE($X^2$ 1_cm²)) and EVAL or EXPAND to get d1UBASE(X^2 1_cm²).	In the command line enter DERIV(UBASE(X^2 1_cm²),X) and EVAL or EXPAND to get d1UBASE(X^2 1_cm²).
UVAL	In the command line enter $X^2\ 1\_cm^2$ and use UVAL to get UVAL(X^2 1_cm²). Enter X and use ∂ to get d1UVAL(X^2 1_cm²).	In the command line enter $X^2\ 1\_cm^2$ and use UVAL to get UVAL(X^2 1_cm²). Enter X and use DERIV to get d1UVAL(X^2 1_cm²).	In the command line enter UVAL(X^2 1_cm²), then X, and use ∂ to get d1UVAL(X^2 1_cm²).	In the command line enter UVAL(X^2 1_cm²), then X, and use DERIV to get d1UVAL(X^2 1_cm²).	In the command line enter $\frac{\partial}{\partial X}$(UVAL($X^2$ 1_cm²)) and EVAL or EXPAND to get d1UVAL(X^2 1_cm²).	In the command line enter DERIV(UVAL(X^2 1_cm²),X) and EVAL or EXPAND to get d1UVAL(X^2 1_cm²).
XOR	X, Y, and then XOR returns Y XOR X. Enter X and use ∂ to get d1XOR(Y,X).	X, Y, and then XOR returns Y XOR X. Enter X and use DERIV to get d1XOR(Y,X).	Enter X XOR Y, then X, and use ∂ to get d1XOR(Y,X).	Enter X XOR Y, then X, and use DERIV to get d1XOR(Y,X).	Enter $\frac{\partial}{\partial X}$(X XOR Y) and EVAL or EXPAND to get d1XOR(Y,X).	Enter DERIV(X XOR Y,X) and EVAL or EXPAND to get d1XOR(Y,X).
XPON	X, and then XPON returns XPON(X). Enter X and use ∂ to get d1XPON(X).	X, and then XPON returns XPON(X). Enter X and use DERIV to get d1XPON(X).	Enter XPON(X), then X, and use ∂ to get d1XPON(X).	Enter XPON(X), then X, and use DERIV to get d1XPON(X).	Enter $\frac{\partial}{\partial X}$(XPON(X)) and EVAL or EXPAND to get d1XPON(X).	Enter DERIV(XPON(X),X) and EVAL or EXPAND to get d1XPON(X).

Function	Enter arguments, then use Function, and then ∂	Enter arguments, then use Function, and then DERIV	Algebraic object Function(X,) then X, then ∂	Algebraic object Function(X,) then X, then DERIV	$\frac{\partial}{\partial X}$(Function(X,)) then EXPAND or EVAL	DERIV(Function(X,),X), then EXPAND or EVAL
ZFACTOR	X, Y, and then ZFACTOR returns ZFACTOR(X,Y). Enter X and use ∂ to get d1ZFACTOR(X,Y).	X, Y, and then ZFACTOR returns ZFACTOR(X,Y). Enter X and use DERIV to get d1ZFACTOR(X,Y).	Enter ZFACTOR(X,Y), then X, and use ∂ to get d1ZFACTOR(X,Y).	Enter ZFACTOR(X,Y), then X, and use DERIV to get d1ZFACTOR(X,Y).	Enter $\frac{\partial}{\partial X}$(ZFACTOR(X,Y)) and EVAL or EXPAND to get d1ZFACTOR(X,Y).	Enter DERIV(ZFACTOR(X,Y),X) and EVAL or EXPAND to get d1ZFACTOR(X, Y).
|	Y, {Y 3 X}, and then | returns 3 X. Enter X and use ∂ to get 3.	Y, {Y 3 X}, and then | returns 3 X. Enter X and use DERIV to get 3.	Enter $Y\|_{Y=3\,X}$, then X, and use ∂ to get 3.	Enter $Y\|_{Y=3\,X}$, then X, and use DERIV to get 3.	Enter $\frac{\partial}{\partial X}(Y\|_{Y=3\,X})$ and EVAL or EXPAND to get d1|(Y,{Y 3 X})	Enter $DERIV(Y\|_{Y=3\,X},X)$ and EVAL or EXPAND to get 3
≤	X, Y, and then ≤ returns X ≤ Y. Enter X and use ∂ to get d1≤(X,Y).	X, Y, and then ≤ returns X ≤ Y. Enter X and use DERIV to get d1≤(X,Y).	Enter X ≤ Y, then X, and use ∂ to get d1≤(X,Y).	Enter X ≤ Y, then X, and use DERIV to get d1≤(X,Y).	Enter $\frac{\partial}{\partial X}(X \le Y)$ and EXPAND or EVAL to get d1≤(X,Y).	Enter DERIV(X ≤ Y,X) and EXPAND or EVAL to get d1≤(X,Y).
≥	X, Y, and then ≥ returns X ≥ Y. Enter X and use ∂ to get d1≥(X,Y).	X, Y, and then ≥ returns X ≥ Y. Enter X and use DERIV to get d1≥(X,Y).	Enter X ≥ Y, then X, and use ∂ to get d1≥(X,Y).	Enter X ≥ Y, then X, and use DERIV to get d1≥(X,Y).	Enter $\frac{\partial}{\partial X}(X \ge Y)$ and EXPAND or EVAL to get d1≥(X,Y).	Enter DERIV(X ≥ Y,X) and EXPAND or EVAL to get d1≥(X,Y).
≠	X, Y, and then ≠ returns X ≠ Y. Enter X and use ∂ to get d1≠(X,Y).	X, Y, and then ≠ returns X ≠ Y. Enter X and use DERIV to get d1≠(X,Y).	Enter X ≠ Y, then X, and use ∂ to get d1≠(X,Y).	Enter X ≠ Y, then X, and use DERIV to get d1≠(X,Y).	Enter $\frac{\partial}{\partial X}(X \ne Y)$ and EXPAND or EVAL to get d1≠(X,Y).	Enter DERIV(X ≠ Y,X) and EXPAND or EVAL to get d1≠(X,Y).

defined derivative because the HP49G can't find the derivative in any syntax. If the cell is yellow, then the HP49G can find the derivative but only using some particular syntax. In such cases you can both make a user defined derivative or use always the particular syntax. The cells coloured green in the same row as the function will show you what the syntax is that you can use to find the derivative. Some times it can get tricky to make the user defined derivative, as we have seen in the case of ! and GAMMA. The tables contain some things that are quite surprising, so when you have time you might want to take a look at them. Mama mia! They were more than I suspected. But I'm through at last, so that we can continue with some more pleasant things. We are going to take a look at some parametric and polar plots on the HP49G, and see what we can do with the stuff we have covered in this part.



We start with the question: Can some body in universe, be it a planet, satellite, comet, or even a stone, be in noncircular, non-elliptic orbit? Before you answer, think again about our built-in pattern recognition machine. We look at the sky, we see circular motions, we develop astronomy based on circular (and elliptical) motion. And so we project "circular" and "elliptical" motions to all bodies in the universe. But let's make a simple example. A planet moves in circular orbit around a start and a satellite in circular orbit around the planet. What will the motion of the satellite look like, seen from some ET that sits above the plain of the ecliptic? In other words, can we give the parametric or polar equations of the motion of the satellite, having the sun as the origin of our coordinates system? Let's try. We assume that the planet moves around the star in distance R from the star and with angular velocity Ω. The satellite moves around the planet in distance d from the planet and with angular velocity ω. The parametric representation of the planet coordinates is:



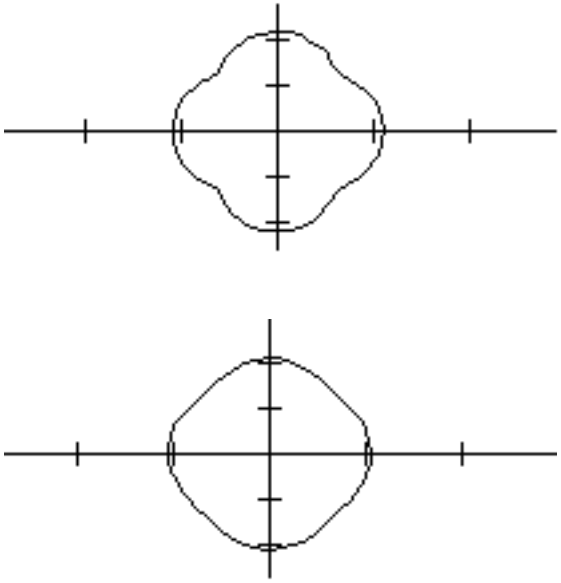
$$X = R\ \cos(\Omega\ t)$$
$$Y = R\ \sin(\Omega\ t)$$

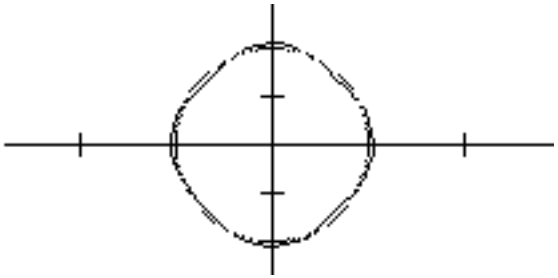
No need to plot that, it is a circle. The parametric representation of the satellite coordinates (having the star as the origin of the coordinates system) is:

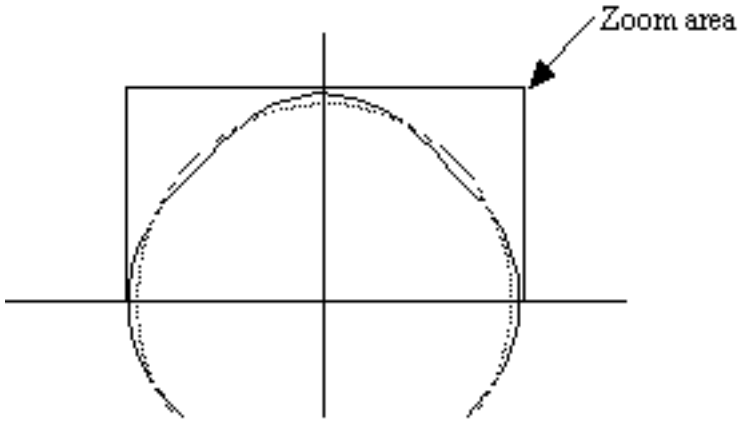
$$x = R\ \cos(\Omega\ t) + d\ \cos(\omega\ t)$$
$$y = R\ \sin(\Omega\ t) + d\ \sin(\omega\ t)$$

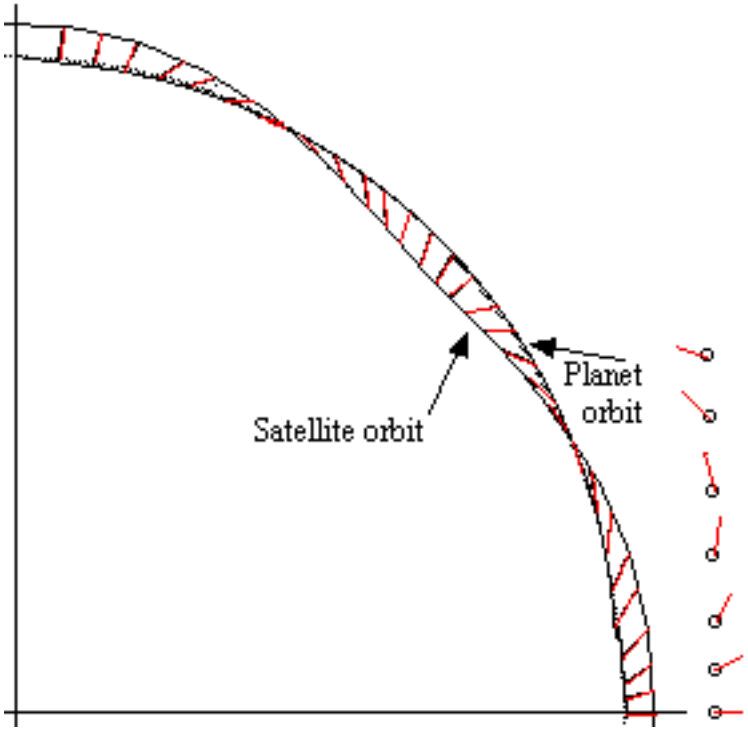
What kind of curve is that? If we assign some values to the variables R, Ω, d, and ω, we can make a parametric plot with the parameter t. For the sake of simplicity we store 1. in R, making thus the distance from the star to the planet to our distance unit in space. The distance d from the satellite to the planet then less than 1. Store .1 in d. Again for simplicity we store 1. in Ω, making the angular velocity of the planet to our angular velocity unit in space. (You get the character Ω by pressing

ALPHA and then → - EQW.) The angular velocity of the satellite on its motion around the planet is in general greater than the angular velocity of the motion of the planet around the sun. Store 5. in ω. (The character ω is ALPHA, → and then EEX.) We are going to make a parametric plot. Go to the PLOT SETUP screen and select plot type Parametric. Enter R COS(Ω t) + d COS(ω t) + i (R SIN(Ω t) + d SIN(ω t)) in the input field EQ:. Enter t as the independent variable (the parameter). Now, in the PLOT WINDOW − PARAMETRIC screen, enter H−View: from −3 to 3, and V−View: from −1.5 to 1.5. Enter Indep Low: 0., and High 6.28. Also enter Step: 1 and activate the option _Pixels. Press ERASE and then DRAW and let the HP49G plot the orbit of the satellite around the star. Wow! Almost a square! Erase the plot, store .05 in d and redraw. Wow^2! Now it is even more like a square! It is interesting to add the orbit of the planet in the same plot. Press ← - F1. Now you are in the equations catalogue screen. Press ADD, to add the parametric equation of the planet motion. The HP49G switches to the EQW and enters automatically XY1(t) =. Delete the unfinished expression XY1(t) = and enter R COS(Ω t) + i R SIN(Ω t). Press ERASE and then DRAW. Because of the limited resolution we can't see very well which the orbit of the planet and which the orbit of the satellite is. We can of course zoom-in to focus on some particular part of the orbits, but let's go the opposite way and enlarge the PICT itself. The PICT is per default 131 pixels wide and 64 pixels high, so let's double its dimensions. Go to the stack, enter #262d and #128d, and press PDIM to resize it to 262 pixels wide and 128 pixels high. Redraw the plot. While the HP49G is plotting, you only see the central part of the plot. Wait until it finishes and then press ← and ◀ to activate scroll mode. In this mode all other graphics functions are deactivated and you can use the arrow keys to scroll around. When you had enough scrolling, press again ← and ◀ to leave scroll mode and return to the normal graphics environment. Here you can also use the arrow keys to move around, but the graphics cursor has to reach the edges of the screen before the scrolling starts. Let's zoom-in now to get an even better impression of what's going on. Use the area indicated in the above picture to zoom in and wait until the plot is done. Move around using the arrow keys to get a first overall impression of the orbits. Though the orbit of the satellite around the star is almost a square, its orbit around the planet is a perfect circle. The picture on the next side demonstrates this. The two orbits are connected with red line segments, which represent the distance d of the

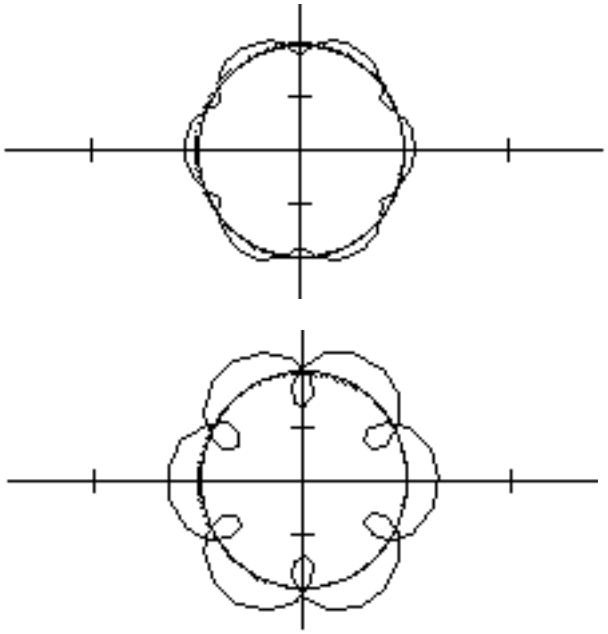








planet to the orbit at several times. At the right part of the picture the relative positions of the planet and the satellite are shown again, to emphasise on the circular motion of the satellite around the planet. The small circle represents the planet, while the red line represents the distance d. The satellite sits on the free end of the red line segments.
If you imagine sitting on the planet and watching the satellite, you will "see" that it has a circular orbit around the planet. Nonetheless it moves in an almost square orbit around the star. Let's try some additional plots with other values for d and ω. First of all, while you are in the graphics environment, press ZOOM, then press twice NXT to go to the third page of the zoom menu, and then press ZLAST to reset the plot to the view ranges before we zoomed in. Then go to the stack, enter #131d and #64d and press PDIM to bring the PICT to its default dimensions. Store .1 in d and 8. in ω. Redraw the orbits. Now you have a six-fold flower (satellite orbit) and a circle (planet orbit). Return to the stack, store .3 in d and redraw to see better how the satellite moves around the planet, while the planet moves around the star.



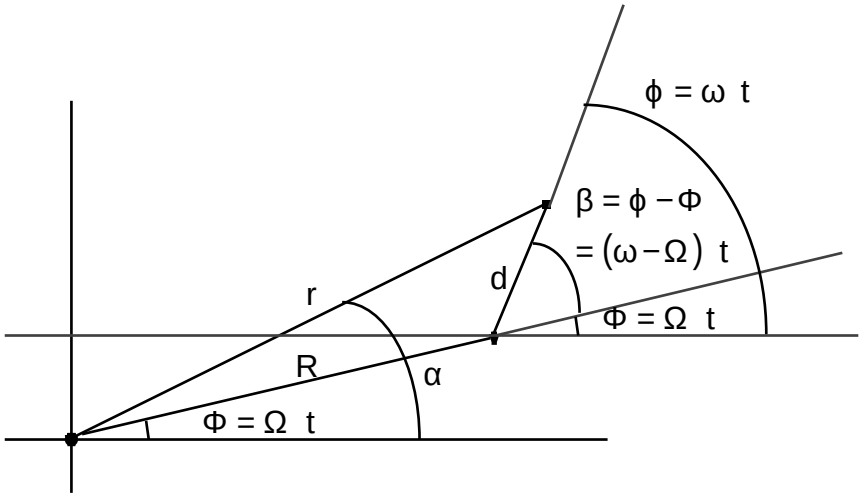
We will try now to convert the parametric form of the orbit of the satellite to a polar form. Enter the list {R Ω d ω} and press PURGE to delete the variables with numeric values. Press VAR to get the variables menu, and press EQ to put the list of the two parametric expressions on the stack. Press HEAD to extract the first expression for the satellite orbit

R COS(Ω t)+d COS(ω t)+i (R SIN(Ω t)+d SIN(ω t))

We could use the function ABS to get the absolute value of the above parametric expression, but then we should add all variables contained in the expression to REALASSUME, because otherwise the HP49G will assume that they are complex, and will return a result containing for example RE(R), IM(R), and so on. Instead of using ABS we find the absolute value "by hand". Switch to real mode (that's important). Press ENTER to make a copy of the parametric expression. Then press RE to get the real part of the expression, and square it, to get SQ(R COS(Ω t)+d COS(ω t)). Press ▶ to swap stack levels 1 and 2. Press IM to get the imaginary part, and square it, to get

$SQ(R\ SIN(\Omega\ t)+d\ SIN(\omega\ t))$. Press [+] to add two squares and then [TCOLLECT] to transform this to $R^2+d^2+2\ d\ R\ COS((\omega-\Omega)\ t)$. Press [√x] to get the square root of the result. The expression:

$$\sqrt{R^2+d^2+2\ d\ R\ COS((\omega-\Omega)\ t)}$$



is the distance from the coordinates origin (star) to the satellite. If you store this in EQ, restore the same values like before for variables R, Ω, d and ω, select plot type Polar, and redraw, you are going to get the same satellite orbit, like in the examples above. Notice that this polar representation doesn't depend on the angle α of the satellite in the coordinates system, in which the star is the origin. It depends on the angle difference between the angle ϕ of the satellite in the coordinates system in which the planet is the origin, and the angle Φ of the planet in the coordinates system in which the star is the origin. If you want to plot $\sqrt{R^2+d^2+2\ d\ R\ COS((\omega-\Omega)\ t)}$ as a polar plot, then you must have t as the independent variable.

We have seen that for some particular values of the variables R, Ω, d and ω, the satellite orbit looks almost like a square. Are there any other particular values of those variables, that make the orbit look like another polygon? We will examine this question in the next parts of this marathon, when we will have covered the additional stuff that is necessary.

Before closing this part, we do another example that shows what complicated curves can be generated by using very simple mechanisms. Consider the simple machine on the bottom of the next page. The (orange metal) arm is connected with the wheel and goes through the metal ring in a distance d from the wheel. The ring itself can freely rotate, but it doesn't move along the x axis. When the wheel spins with angular velocity ω, what will be the curve that the end of the (orange metal) arm will create? In order to answer this (and plot this) we must find some analytic form of the coordinates of the end of the arm, or take the numeric way. In this case it is easy to follow the analytic way. We need to find the angle α, because if we have it, then we can use it to find the lengths of the projections of the arm on the x and on the y axis. This angle can be found by using the formula:

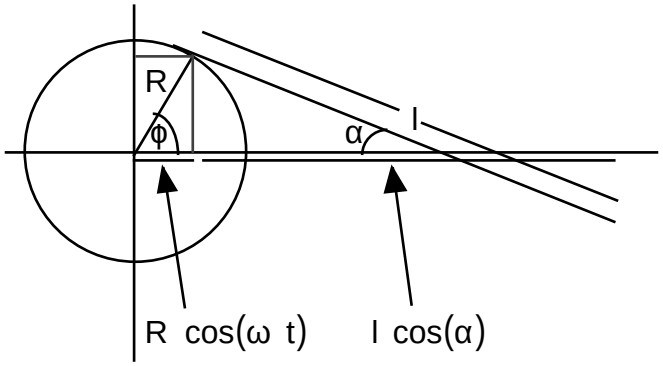
$$\alpha = atan\left(\frac{R\ sin(\omega\ t)}{d+R-R\ cos(\omega\ t)}\right)$$

Having this angle we can find the x coordinate of the end of the metal arm. Enter:

$$ATAN\left(\frac{R\ SIN(\omega\ t)}{d+R-R\ COS(\omega\ t)}\right)$$

and store the expression in variable α. Now enter $R\ COS(\omega\ t)+l\ COS(\alpha)$. In the EQW select the sub expression $l\ COS(\alpha)$ and expand. Press [ENTER] to put the expression for the x coordinate on the stack. The expression is:

$$R\ \mathrm{COS}(\omega\ t)+\sqrt{\left(R^4\ \mathrm{COS}(\omega\ t)^2-\left(2\ R^4+2\ d\ R^3\right)\ \mathrm{COS}(\omega\ t)+R^4+2\ d\ R^3+d^2\ R^2\right)\ \mathrm{SIN}(\omega\ t)^2+R^4\ \mathrm{COS}(\omega\ t)^4-\left(4\ R^4+4\ d\ R^3\right)\ \mathrm{COS}(\omega\ t)^3+\left(6\ R^4+12\ d\ R^3+6\ d^2\ R^2\right)\ \mathrm{COS}(\omega\ t)^2-\left(4\ R^4+12\ d\ R^3+12\ d^2\ R^2+4\ d^3\ R\right)\ \mathrm{COS}(\omega\ t)+R^4+4\ d\ R^3+6\ d^2\ R^2+4\ d^3\ R+d^4}\ /\ \left(R^2\ \mathrm{SIN}(\omega\ t)^2+R^2\ \mathrm{COS}(\omega\ t)^2-\left(2\ R^2+2\ d\ R\right)\ \mathrm{COS}(\omega\ t)+2\ R^2+2\ d\ R+d^2\right)$$

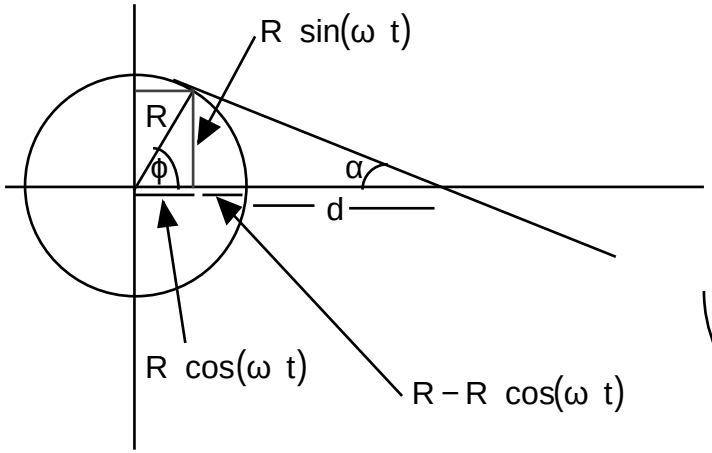


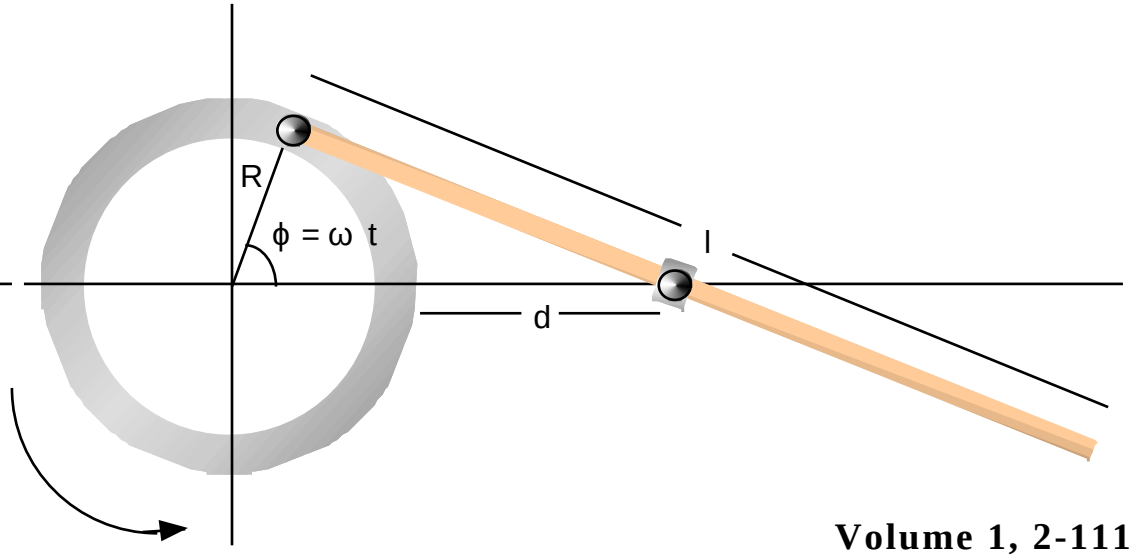
Alone the huge sub expression under the square root is reason for making us to lose any interest to handle the problem. It wouldn't fit the width of this page written in one line. But the HP49G is a much more powerful companion than it might look. Press ▼ to take the whole expression in the EQW. Use the arrow keys to select the sub expression under the square root. Press COLLECT and then EVAL. The HP49G converts this sub expression to:

$$\left(R\ \mathrm{COS}(\omega\ t)-(R+d)\right)^2\ \left(R^2\ \mathrm{SIN}(\omega\ t)^2+R^2\ \mathrm{COS}(\omega\ t)^2-\left(2\ R^2+2\ d\ R\right)\ \mathrm{COS}(\omega\ t)+R^2+2\ d\ R+d^2\right)$$

$R^2\ \mathrm{SIN}(\omega\ t)^2$, press ◄ and then ► to extend the selection to $R^2\ \mathrm{SIN}(\omega\ t)^2$. Press TRIG to convert the selection to R^2. Using the same technique select the sub expression $2\ R^2+2\ d\ R$ and collect it to $(R+d)\ R\ 2$. Select the sub expression $R^2+2\ d\ R+d^2$ and collect it to $(R+d)^2$. Now we move on to the denominator. Use the arrow keys to select the sub expression $R^2\ \mathrm{SIN}(\omega\ t)^2+R^2\ \mathrm{COS}(\omega\ t)^2$ of the

and this brings our hopes back that we will somehow come to an end ;-). While in the EQW select the term





denominator and press again TRIG to convert it to R^2. Select the expression $2\ R^2 + 2\ d\ R$ and collect it to $(R+d)\ R\ 2$. Select the expression $R^2 + 2\ d\ R + d^2$ of the denominator and collect it to $(R+d)^2$. Press ENTER to put the whole expression on the stack. Now it looks much better:

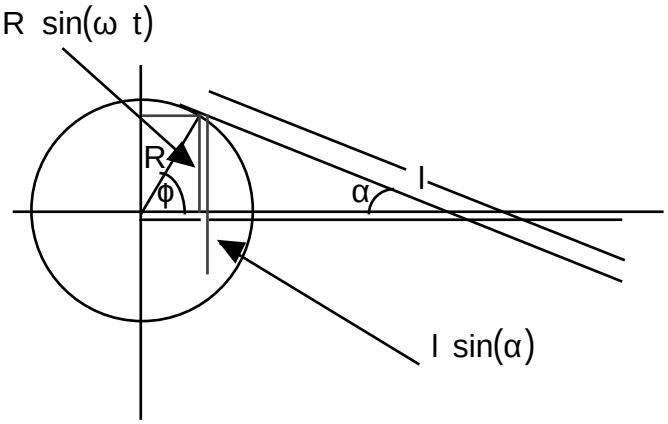
$$R\ COS(\omega\ t) + \frac{I\sqrt{(R\ COS(\omega\ t)-(R+d))^2}}{\sqrt{(R^2-(R+d)\ R\ 2\ COS(\omega\ t)+(R+d)^2)}}$$

Store this in variable x (small letter).

Now we are going to find an expression for the coordinate y. Enter $R\ SIN(\omega\ t) - I\ SIN(\alpha)$. Press ▼ to take the expression in the EQW. In the EQW select the sub expression $-I\ SIN(\alpha)$ and expand. Press ENTER to put the expression for the y coordinate on the stack. The expression is:

$$R\ SIN(\omega\ t) + \frac{I\ R\ SIN(\omega\ t)\sqrt{\begin{matrix}(R^4\ COS(\omega\ t)^2-(2\ R^4+2\ d\ R^3)\ COS(\omega\ t)+R^4+2\ d\ R^3+d^2\ R^2)\ SIN(\omega\ t)^2+\\ R^4\ COS(\omega\ t)^4-(4\ R^4+4\ d\ R^3)\ COS(\omega\ t)^3+(6\ R^4+12\ d\ R^3+6\ d^2\ R^2)\ COS(\omega\ t)^2-\\ (4\ R^4+12\ d\ R^3+12\ d^2\ R^2+4\ d^3\ R)\ COS(\omega\ t)+R^4+4\ d\ R^3+6\ d^2\ R^2+4\ d^3\ R+d^4\end{matrix}}}{\begin{matrix}(R^3\ COS(\omega\ t)-(R^3+d\ R^3))\ SIN(\omega\ t)^2-R^4\ COS(\omega\ t)^4-(3\ R^3+3\ d\ R^2)\ COS(\omega\ t)^2+\\ (3\ R^3+6\ d\ R^2+3\ d^2\ R)\ COS(\omega\ t)-(R^3+3\ d\ R^2+3\ d^2\ R+d^3)\end{matrix}}$$

which is again a monster that the HP49G can tame. select the whole sub expression under the square root and collect it. The HP49G fights bravely and manages to collect the sub monster, errh, I mean sub expression to



$$\left(R\ COS(\omega\ t)-(R+d)\right)^2\ \begin{matrix}\text{'}\sqrt{R^2\ SIN(\omega\ t)^2+R^2\ COS(\omega\ t)^2}\text{'}\\ (2\ R^2+2\ d\ R)\ COS(\omega\ t)+\\ R^2+2\ d\ R+d^2\end{matrix}$$

While in the EQW select the sub expression $R^2\ SIN(\omega\ t)^2+R^2\ SIN(\omega\ t)^2$, and press TRIG to convert the selection to R^2. Select the sub expression $2\ R^2 + 2\ d\ R$ and collect it to $(R+d)\ R\ 2$. Select the sub expression $R^2 + 2\ d\ R + d^2$ and collect it to $(R+d)^2$. Select the whole denominator and collect to get the sub expression:

$$\left(R\ COS(\omega\ t)-(R+d)\right)\ \begin{matrix}\text{'}\sqrt{R^2\ SIN(\omega\ t)^2+R^2\ COS(\omega\ t)^2}\text{'}\\ (2\ R^2+2\ d\ R)\ COS(\omega\ t)+\\ R^2+2\ d\ R+d^2\end{matrix}$$

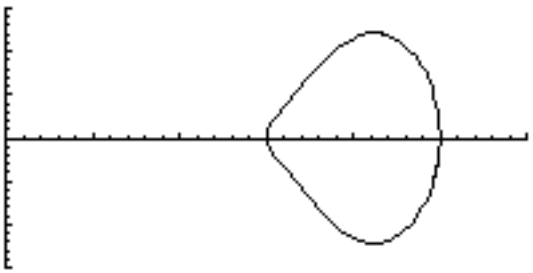
Press TRIG to convert $R^2\ SIN(\omega\ t)^2+R^2\ SIN(\omega\ t)^2$ to R^2. Then collect $2\ R^2+2\ d\ R$ to $(R+d)\ R\ 2$, and $R^2+2\ d\ R+d^2$ to $(R+d)^2$. The expression for the y coordinate of the end of the metal arm looks now also much better. We still can't tell that it is an "easy" expression, but comparing it to what we started with, it is simpler:

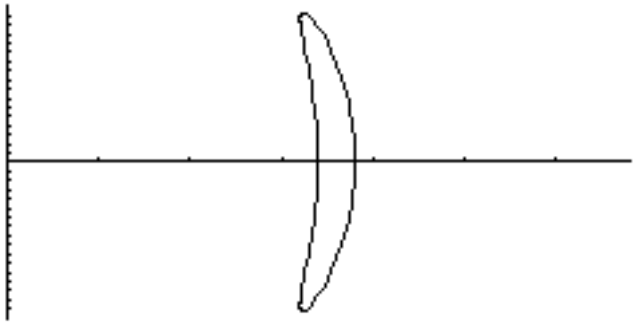
$$R\ SIN(\omega\ t)+\frac{l\ R\ SIN(\omega\ t)\ \sqrt{(R\ COS(\omega\ t)-(R+d))^2}}{\left(R\ COS(\omega\ t)-(R+d)\right)\ \sqrt{\left(R^2-(R+d)\ R\ 2\ COS(\omega\ t)+(R+d)^2\right)}}$$

Store it in y (small letter).

Before we do the plot for this parametric function, some words about the math capabilities of the HP49G. As you saw the expressions we deal with in this example are way not "easy". Nonetheless the HP49G is a great help. Consider for example how long it would take to do the same by hand. It is not only a matter "knowing how to" but also a matter of mistakes that one could do, for example because of forgetting a power, and the like. The calculator is no substitute for our thinking (fortunately) but rather a help for freeing us from the "dirty work" and letting us concentrate on the important things. Also notice that we didn't simply said, "HP49G think for me and simplify this expression". We considered the expressions, we decided *what* would be best to do, and tried it out. Of course, often our decision will not bring any advantages when it comes to simplifications. But with the HP49G this is not bad at all. Quite the contrary, it is very good. Because this way, after a certain amount of experience, we start "knowing" what some particular operation will give us, *before* we carry it out. Not exactly each and every terms of the result, but rather the overall shape of the result. Pattern recognition?

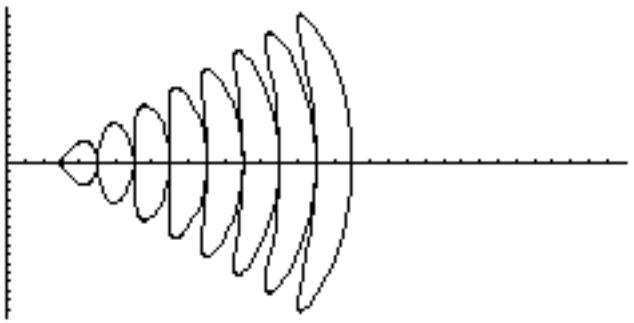
Curious as we are, we want to see what the above parametric function looks like. In order to plot it, we must store some values in the variables R, d, l, and ω. Store 1. in R, 1. in d, 4. in l, and 1. in ω. In the variables menu, press x, enter i, press y, then ⊠ and then +. Store the resulting expression in EQ. In the PLOT SETUP screen, set plot type Parametric, and set independent variable t. In the PLOT WINDOW − PARAMETRIC screen, set H−View: from 0. to 6. and V−View: from −1.5 to 1.5. Set Indep Low: to 0.. When the input field High: is selected, enter the sequence 2 π → NUM, to enter the numeric approximation of 2 π. Finally, enter the sequence 2 π 50 / → NUM in the input field Step:. Press ERASE and then DRAW. You get the "drop" at the right. Let's try another plot but now with a much bigger l. Store 18. in l, set H−View: from 0. to 34. and V−View: from −8.5 to 8.5., and erase and draw again. This time you get a banana. What are that curves between l = 4. and l = 18.? Let's see. We will do a small program that draws the curves inside the above limits for a step of 2., that is for l = 4., l = 6., up to l = 18.. Enter the program:





```
<<
  ERASE DRAX
  4. 18.
  FOR I
    I 'l' STO DRAW
  2. STEP
>>
```

Press EVAL to let it run and watch how the generated curves change shape as I gets bigger and bigger. When the program finishes, press [arrow-left] to go the graphics environment, turn the HP49G 90° in clockwise direction, and see how drops and bananas can be combine to give us a pine.



This example of a parametric function shows also that interesting curves can be produced out of simple mechanisms. It would be a good exercise to think about some "machines", with connected wheels and arms and axes and gear, and try to use the HP49G for plotting the curves of some point of the moving parts of the machine. The resulting parametric expressions are most of the time rather "inconvenient" but with the brave HP49G on your side you will be able to defeat most of the monsters. But some of them, often the most "easy looking", will prove very very "noncooperative". Like Trabakoulas said, "Pattern recognition can get quite dangerous, when it comes to easy and difficult. Exact examination is far better."

We finish this part, and I think that it is time for me to go sleep about one week. I see curves and lines instead of pines, which means that I need a good old Lagavulin to return to reality. Take care and 'till next time.

Parametric greetings,
Nick.

Basic Calculus with the HP49G - Volume 1 - Part 3

Hoi zämme![2]

The last part of this marathon has been quite…, well, marathonial. After that we will surely enjoy a part with much stuff to play with. As Trabakoulas says, "Learning is best made in the game". So, while other people are playing "how to be become of the master of the world", we ignore that totally uninteresting (and questionable) games, and we play again, what we play best: How much maths is possible on the HP49G?. From what we've seen until now, the answer to this question is "much, much more than we could imagine considering only the size of the machine. And who knows what is yet to come.

Until now we focused on functions of a single variable. But a function can have more than one variables. Consider the function $f(u,v,w)$, which depends on the variables u, v, and w. If we consider any two of these variables, say v and w, as quantities, the values of which we "hold" constant, then the function $f(u,v,w)$ can be considered as a function of a single variable, namely of u. We "freeze" both v and w at some arbitrary values, and consider how the function behaves when u varies. That means, we consider the *partial* dependency of $f(u,v,w)$ on variable u. Then, the *partial* derivative of $f(u,v,w)$ for u is $\frac{\partial f}{\partial u}$. Actually the "curly" ∂ is used in mathematics for such partial derivatives. The normal derivative is denoted with "normal" d, i.e. $\frac{df}{dx}$. But because the HP49G considers any variable that doesn't depend on the differentiation variable as constant, it uses only the symbol ∂. Let's do some examples.

Enter $X^3+7\ X^2\ Y+3\ X\ Y^5-5\ Y^6$ and press ENTER to make a copy of this expression. Let's find the partial derivative for X. Enter X and then press ∂ to get $3\ X^2+Y\ 7\ 2\ X+3\ Y^5$. Expanding this you get $3\ X^2+14\ Y\ X+3\ Y^5$. Press ▶ and let's find the partial derivative for Y. Enter Y and press DERIV to get $7\ X^2+3\ X\ 5\ Y^4-5\ 6\ Y^5$. Expand this to get $7\ X^2+15\ Y^4\ X-30\ Y^5$. As you can see the HP49G has ordered the powers of X in descending order. If you want to order for Y powers, enter Y and press REORDER to get $-(30\ Y^5-15\ X\ Y^4-7\ X^2)$. If you don't like the minus sign in front of the whole expression, then press FDISTRIB, to get $-(30\ Y^5)+15\ X\ Y^4+7\ X^2$. If you want ascending ordering, then set flag -114 and expand. The result is $-(30\ Y^5-15\ X\ Y^4-7\ X^2)$, that means in ascending order of powers of X. If you now enter Y and press REORDER again, then you get $7\ X^2+15\ Y^4\ X-30\ Y^5$, in which the powers of Y are sorted in ascending order. The command EXPAND orders the powers according to the flag -114. It prefers automatically the variable that is lower in alphabetical order. That means, if you have an expression with variables A and B, then it will try to order for A according to the state of flag -114. The command REORDER allows you to select which variable's powers will be used for ordering. Clear flag -114 now.

Let's do some more examples. Enter:

$$\frac{\partial}{\partial X}\left((X+Y)^3\ e^{X\,Y}\right)$$

and make a copy of the expression because we will need it more than once. If you press EXPAND now, you will get $\left(Y\ X^3+\left(3\ Y^2+3\right)\ X^2+\left(3\ Y^3+6\ Y\right)\ X+Y^4+3\ Y^2\right)\ e^{X\,Y}$. Press ▶ to bring the expression:

$$\frac{\partial}{\partial X}\left((X+Y)^3\ e^{X\,Y}\right)$$

[2] Swiss "Hi everybody", pronounced "Hoi tsama", or something like that. The letters of the alphabet are simply not enough to represent this singing language accurately.

on stack level 1, and make another copy of it. Now press EVAL. The result this time is $3\ (X+Y)^2\ e^{X\,Y}+(X+Y)^3\ Y\ e^{X\,Y}$. This shows that EXPAND and EVAL are different things. They don't always return the same results when applied on algebraic expressions. The last result demonstrates also another fact. Suppose that for some reason you want to expand the factor $3\ (X+Y)^2$ of the term $3\ (X+Y)^2\ e^{X\,Y}$, but leave the rest of the expression unchanged. Interactively you would take the whole expression in the EQW, select $3\ (X+Y)^2$ using the arrow keys, and expand it. But there is no built-in programmable command for doing the same from a program. We can't apply some command, like EXPAND or COLLECT, to a *part* of an expression programmatically. One of the features I miss most on the HP49G is exactly this. The next pages will demonstrate an imperfect way to achieve this, which I nonetheless believe that it is good enough for generation of ideas.

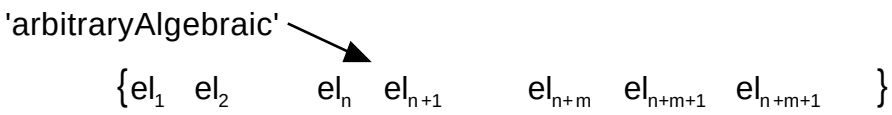
Having the commands → LST and → ALG, it is not difficult to make programs that apply some commands to a part of an expression. Let's examine these commands. With the expression $3\ (X+Y)^2\ e^{X\,Y}+(X+Y)^3\ Y\ e^{X\,Y}$ on stack level 1, press →LST to get:

```
1: { 3  X  Y  +  2  ^  EXP  X  Y  +  3  ^  Y
     X  Y  EXP  + }
```

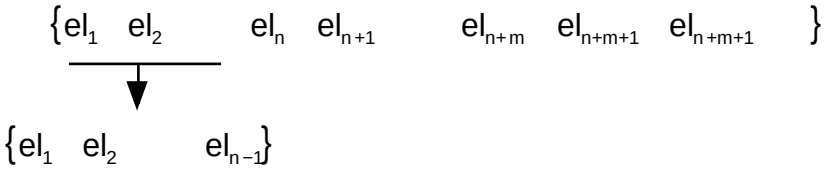
The first 7 list elements represent the expression $3\ (X+Y)^2$ that we want to expand. Make a copy of the list. Enter 1, then 7, and press SUB to create the sub list { 3 X Y + 2 ^ }. Now press → ALG to convert the list to $3\ (X+Y)^2$. Expand this expression to get $3\ X^2+6\ X\ Y+3\ Y^2$ and press →LST to convert it to the list to { 3 X 2 ^ 6 Y X + 3 Y 2 ^ + }. Press ▶ to bring the big list to stack level 1. Now we will create the sub list of all elements that we didn't use, i.e elements 8 to last. Enter 8 and press OVER and then SIZE, to get 25.. Press SUB to get { X Y EXP X Y + 3 ^ Y X Y EXP + }. Press + to add the the expanded sub expression to the list on stack level 1. Now press → ALG to get $\left(3\ X^2+6\ X\ Y+3\ Y^2\right)\ e^{X\,Y}+(X+Y)^3\ Y\ e^{X\,Y}$.

In general the method looks like this:

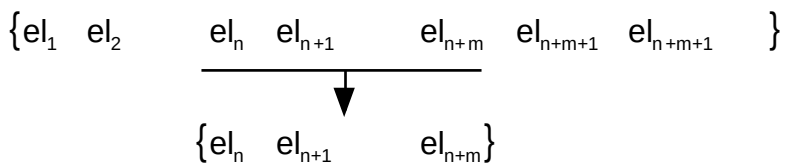
Step 1: Convert the algebraic object to its equivalent RPL list using the command → LST.

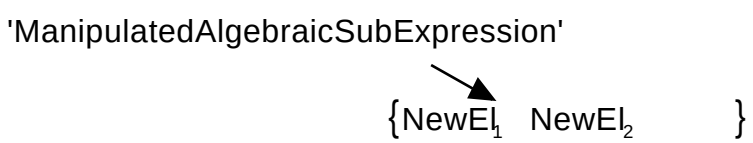


Step 2: Create the sub list that contains elements 1 up to n-1, where n is the first element that belongs to the sub expression which we want to manipulate.



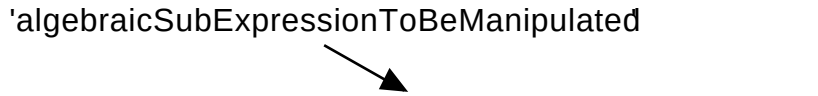
Step 3: Create the sub list that contains elements n up to n+m, where n is the first element and n+m is the last element of the sub expression which we want to manipulate.





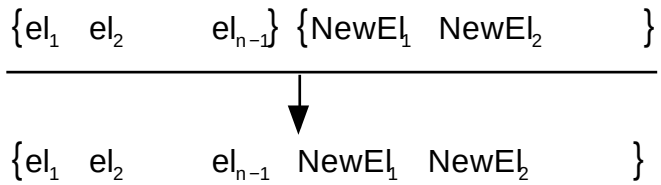
Step 4: Use → ALG to convert the list $\{el_n \quad el_{n+1} \quad el_{n+m}\}$ to its corresponding algebraic object.

Step 5: Apply the algebraic manipulation (EXPAND, COLLECT, etc.) to 'algebraicSubExpressionToBeManipulated'.

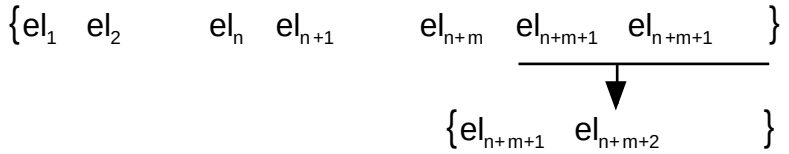


Step 6: Convert 'ManipulatedAlgebraicSubExpression' to the list $\{newEl_1 \quad newEel_2 \quad\}$ with → LST.

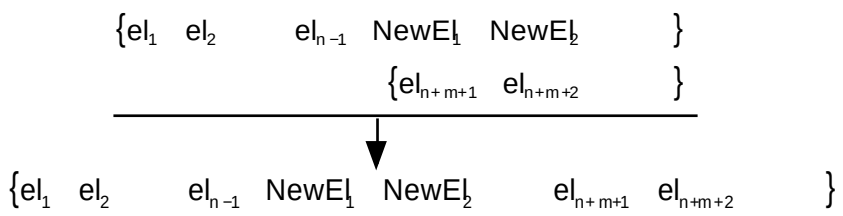
Step 7: Add $\{NewEl_1 \quad NewEl_2 \quad\}$ to the list $\{el_1 \quad el_2 \quad el_{n-1}\}$.



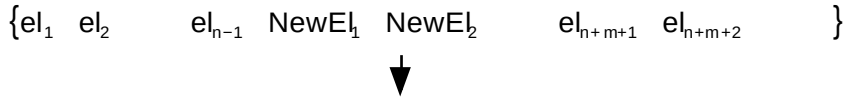
Step 8: Create the sub list that contains elements n+m+1 up to the last, where n+m is the last element that belongs to the sub expression which we want to manipulate.



Step 9: Add the sub list $\{el_{n+m+1} \quad el_{n+m+2} \quad\}$ to the list $\{el_1 \quad el_2 \quad el_{n-1} \quad NewEl_1 \quad NewEl_2 \quad\}$.



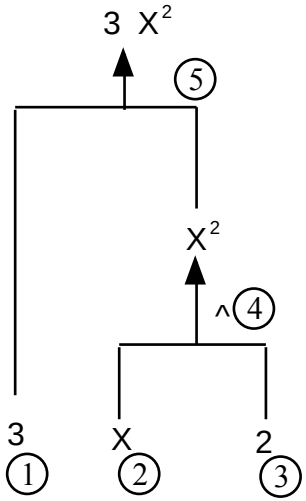
Step 10: Use → ALG to convert the last list to an algebraic object.



This is indeed a very easy thing to program. We will make a program that takes an algebraic object from stack level 2, and a list from stack level 1. The list contains will have three items. The first item is the manipulation that we want to apply to a sub expression of the algebraic object. It can be a command, like EXPAND, COLLECT, etc. But it can also be function, like SIN, COS, etc. It can be even a program, provided that the program takes exactly one algebraic object as input and returns exactly one algebraic as output. The second item is the position of the first RPL list element that belongs to the sub expression that we want to manipulate. The third item is the position of the last RPL list element that belongs to the sub expression that we want to manipulate. Here is the program listing:

```
<<
  SWAP →LST
  → specs expr
  <<
    expr 1 specs 2 GET          @Create first sub list
    1 - SUB
    expr specs 2 GET            @Create sub list of alg.
    specs 3 GET SUB             @to be manipulated.
    →ALG specs HEAD             @Apply command. Add to
    EVAL →LST +                 @first sub list.
    expr 3 GET 1 +              @Create third sub list,
    OVER SIZE SUB +             @add to the rest
    →ALG                        @Convert to algebraic
  >>
>>
```

This is the program APLAT (APpLy AT, name borrowed from Mathematica). To use this program we must first see how we number the elements of an algebraic expression, in order to be able to provide the program with correct information. If you didn't drop the last result of page 3-2, then stack level 1 must contain the expression $\left(3\ X^2+6\ X\ Y+3\ Y^2\right)\ e^{X\,Y}+(X+Y)^3\ Y\ e^{X\,Y}$. Let's see how the elements of this expression are numbered. We start numbering at the first element excluding parentheses, as parentheses are not included in the elements of the corresponding RPL list. (This is a result of the RPL method, which doesn't need any parentheses at all.) We count first the operands (arguments) and then the functions that act upon the operants. For example in the above expression, we start with $3\ X^2$. The two arguments 3 and X^2 are combined by (multiplication). The argument 3 has the number 1. The argument X^2 is itself an expression, so we can't give it the number 2. This expression contains the two arguments X and 2, which are combined by ^ (power). So, X is number 2, the power 2 is number 3, the power function ^ is number 4, and the multiplication of 3 with X^2 is number 5. The picture on the top of the right column illustrates this. That means, when we want to somehow manipulate the sub expression $3\ X^2$, we



have to give the numbers 1 and 5 to our program.

The sub expression $3\ X^2+6\ X\ Y+3\ Y^2$ includes elements 1 to 17. Let' try our program. We will collect the sub expression $3\ X^2+6\ X\ Y+3\ Y^2$ but will leave the rest unchanged. Since the expression $\left(3\ X^2+6\ X\ Y+3\ Y^2\right)\ e^{X\ Y}+(X+Y)^3\ Y\ e^{X\ Y}$ is on stack level 1, we don't need to type it again. Enter the list {COLLECT 1 17} and press APLAT. The result is $(X+Y)^2\ 3\ e^{X\ Y}+(X+Y)^3\ Y\ e^{X\ Y}$. It worked!

But there are shadows on our way. The corresponding RPL list of an algebraic object contains no parentheses, but the algebraic object itself can contain parentheses. To understand the problem better we consider a simple example. Enter the algebraic object $A+B+C$. What problems can this simple expression bring? Well, set flag -53 (to let all parentheses be shown), press ◀ and then ▼ to edit the expression in the command line. The expression is shown as '(A + B) + C'. There are invisible parentheses in this simple object!!! What does this mean for us? First of all, suppose we have $A+B+C$ and we want to do something with the sub expression $B+C$. This means that we should give the program APLAT the numbers 3 and 5, according to our numbering system. Press ENTER to put the expression to the stack, and enter the list {EXPAND 3 5}. Press APLAT to (try to) expand the sub expression $B+C$. Of course in this example you can't expand anything, but this is only for demonstration purposes. The HP49G errors out with Bad Argument Type and leaves 'Invalid Expression' on stack level 1. What happened here? Press UNDO and then drop the list. Now the object $A+B+C$ must be back on stack level 1. Press →LST. The result is the list {A B + C +}. Elements 3 to 5 build up the sub list {+ C +}, out of which the program APLAT tried to make a sub expression. This list is of course not the corresponding RPL list of the sub expression $B+C$. Can we build up the sub expression $B+C$ out of the (current) algebraic object $A+B+C$? What numbers do we have to give to APLAT to achieve this? The answer is: The current inner structure of the algebraic object $A+B+C$ makes this task impossible. In reality the object, as it is now, is $(A+B)+C$. In this object the sub expression $B+C$ doesn't exist!!! Don't confuse this with the well known (and almost spontaneous) recognition of the pattern $(A+B)+C=A+(B+C)$, which "automatically" implies that our object is equivalent to $A+(B+C)$, in which the sub expression $B+C$ does exist. The calculator doesn't have such "spontaneous" cognition capabilities. Before we go further take a look at the following table:

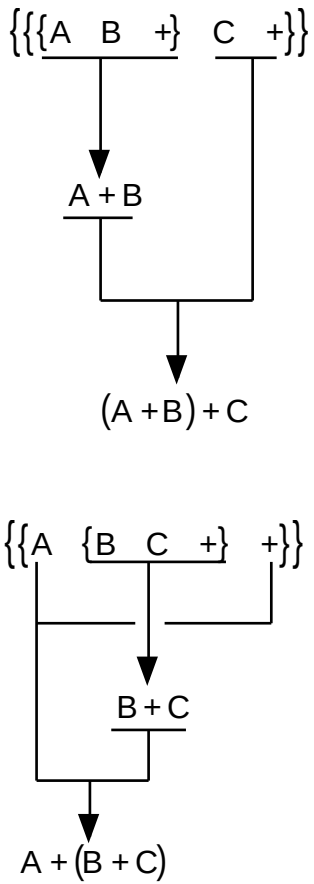
Algebraic Object	RPL List	Sub expression $B+C$
$(A+B)+C$	{A B + C +}	Doesn't exist
$A+(B+C)$	{A B C + +}	Elements 2 to 4

Though the two objects $(A+B)+C$ and $A+(B+C)$ are mathematically equivalent, for the HP49G they are different!!

In this example, if you expand the current version of $A+B+C$, the calculator alters its inner structure from $(A+B)+C$ to $A+(B+C)$, i.e. from {A B + C +} to {A B C + +}. Press EXPAND, then ◀, and then ▼, to see the object in the form $A+(B+C)$. Press ENTER to put it back to the stack. Now of course you can enter the list {EXPAND 2 4} and press APLAT. But now the sub expression $A+B$ has become unavailable for partial manipulations. And the big problem is that there are simply too many internal rules that the HP49G uses in order to decide which sub expressions to put in parentheses. (And these rules are kept secret by the makers.) Wanting to make a program that will run successfully in any possible case, is like wanting to do reverse engineering on the whole CAS of the HP49G, and then make a program that "knows" all rules and all possible way to rearrange an expression. This would be also similar to mapping all properties like commutativity, associativity etc., to a single program… errrh, who's gonna do that? ;-)

From the above paragraphs we see that it is not so easy to specify a mathematically valid sub expression of an expression on the HP49G. And since there are no built in commands that can give us some particular (mathematically valid) sub part of a given expression, I think we have a major disadvantage. We will return to these fields some day, in a special marathon run, but for now we take a look at three small programs that can bring us some light when we want to know exactly, what the internal structure of an algebraic object looks like.

First of all we make a program that converts a given algebraic object to a *structured* list. We continue using the example $A+B+C$ in its two different versions, $(A+B)+C$ and $A+(B+C)$. When the program is fed with $(A+B)+C$, it should return the list $\{\{\{A\ B\ +\}\ C\ +\}\}$, in which the sub expression $(A+B)$ (in the invisible parentheses) is put in a sub list, which itself is element of another sub list that contains the "rest". But when the program is fed with $A+(B+C)$, it should return the list $\{\{A\ \{B\ C\ +\}\ +\}\}$, in which the sub expression $(B+C)$ (in the invisible parentheses) is put in a sub list, which itself is element of another sub list that contains the "rest". The structured list that the program returns is one representation of the algebraic tree that builds up the algebraic object. Consider the following code:




```
<<
  IF
    DUP TYPE 9. ==
  THEN
    1. →LIST
  END
  1.
  <<
    IF
      DUP TYPE 9. ==
    THEN
      OBJ→ SWAP 1. + →LIST
      ALG→TREE
    END
  >>
  DOSUBS
>>
```

This is the program ALG → TREE that comes with this document. It takes one algebraic object from the stack and it returns its algebraic tree list. As you can see it calls itself over and over again, until the whole object has been processed. Let's try it. Enter:

$$Y\ SIN(X)^2+\left(Y-\frac{1}{2}\right)COS(X)^2$$

and make one copy of this algebraic object. Now press ALG→TREE. The result is the list:

{{{Y {{X SIN} 2 ^} } {{Y {1 2 /} −} {{X COS} 2 ^} } +}}

If you imagine all but the outermost brackets away, then you have the list that the command → LST would return. Press ▶ and then →LST. You get the list:

{Y X SIN 2 ^ Y 1 2 / − X COS 2 ^ +}

which indeed is equivalent to the tree list, up to additional structuring. The tree list, so to speak, includes the (invisible) parentheses that are present to algebraic objects, while the RPL list is the command sequence that produces exactly the same algebraic object.

Since we have a the program ALG → TREE, we are inclined to program the opposite, TREE → ALG. If we could somehow flatten the tree list, we would get the RPL list of the algebraic object, which we can convert to an algebraic object using → ALG. Here we can use the program FLATTEN again, which we made in the Complex Numbers Marathon:

```
<<
  1. CF
  1.
  <<
    IF
      DUP TYPE 5. ==
    THEN
      OBJ→ DROP
      1. SF
    END
  >>
  DOSUBS
  IF
    1. FS?
  THEN
    FLATTEN
  END
>>
```

As you can see the program FLATTEN is one of those that call themselves until some condition (here: flag 1 is clear) is true. Press ▶ and then FLATTEN, to convert the tree list to the RPL list. Now you can press → ALG to get the original algebraic object:

$$Y\ \mathrm{SIN}(X)^2+\left(Y-\frac{1}{2}\right)\mathrm{COS}(X)^2$$

The program TREE → ALG can be easily made:

```
<<
  FLATTEN →ALG
>>
```

Sometimes programs are easy ;-)

With the algebraic expression:

$$Y\ \mathrm{SIN}(X)^2+\left(Y-\frac{1}{2}\right)\mathrm{COS}(X)^2$$

on stack level 1, press ALG→TREE to get the algebraic tree list again. Now, press TREE→ALG to get the algebraic object out of the tree list.

Using the tree list we can introduce another numbering system for sub expressions. The first one that we used, was simply the range of position numbers in the RPL list of an algebraic object. For example the sub expression:

$$Y-\frac{1}{2}$$

of the algebraic object:

$$Y\ \mathrm{SIN}(X)^2+\left(Y-\frac{1}{2}\right)\mathrm{COS}(X)^2$$

can be specified by the range 7…11 in the corresponding RPL list:

{ Y X SIN 2 ^ Y 1 2 / − X COS 2 ^ + }

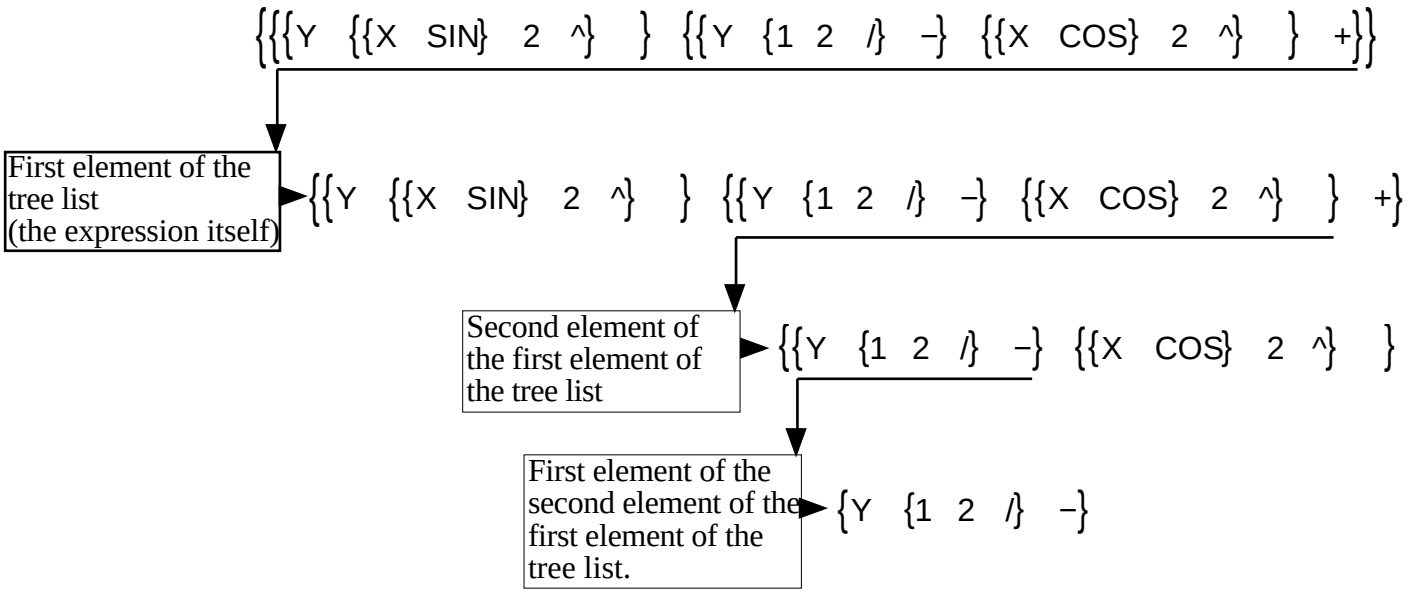
The same sub expression can be specified by giving the position numbers of the sub lists, sub sub lists, and so on, of the corresponding tree list. The tree list of the above expression was:

{{{Y {{X SIN} 2 ^} } {{Y {1 2 /} −} {{X COS} 2 ^} } +}}

In this list the sub expression:

$$Y - \frac{1}{2}$$

is the 1st. element of the 2nd. element of the 1st. element of the tree list. That means that we can specify this sub expression by giving the "coordinates list" {1 2 1}. If we put away the first coordinate which is always 1 (i.e. the expression itself) then we can specify the sub expression with {2 1}. Programming the extraction of sub expressions from algebraic objects is then easy:




```
<< DEPTH PICK3 ALG→TREE HEAD
   → alg part depth tree
  << tree part 1. << GET >>
    IFERR
      DOSUBS
    THEN
      DEPTH depth - 2 + DROPN
      alg part
      "Bad Part Specification" DOERR
    ELSE
      IF DUP TYPE 5. ==
      THEN TREE→ALG
      END
    END
  >>
>>
```

This is the program GETSUBEX that comes with this document. It takes an algebraic object and a list with the sequence of sub parts that specifies the sub expression that we want. This list must not contain the first 1, which is the algebraic object itself. The program uses ALG → TREE to convert the algebraic object to a tree list. Then it takes the parts specified by the "coordinates list". If it fails because of wrong part specification, it cleans up the stack and exits giving you a message Bad Part Specification. Else, you get the corresponding sub expression. In case the part specification specifies something different than a sub expression (a function, a variable, etc.) you get the corresponding object alone. (Not in an algebraic object). Let's give it a try.

Enter again:

$$Y\ \ SIN(X)^2 + \left(Y - \frac{1}{2}\right)\ COS(X)^2$$

then $\{2\ \ 1\}$, and press GETSUBEX to get:

$$Y - \frac{1}{2}$$

The program GETSUBEX can be used to implement partial algebraic object manipulation.

```
<<
  → alg manipart
  <<
    alg DUP
    manipart 2. GET
    GETSUBEX DUP
    manipart HEAD EVAL
    2. →LIST
    ↑MATCH DROP
  >>
>>
```

This is the program APLSUBEX. It takes an algebraic object from stack level 2, and a list from stack level 1. The list has two elements. The first is the command or program that has to be applied on some sub expression of the algebraic object. The second is a list that specifies the sub expression. Try it? OK! Enter:

$$Y\ \ SIN(X)^2 + \left(Y - \frac{1}{2}\right)\ COS(X)^2$$

We will linearise the sub expression

$$SIN(X)^2.$$

This is sub expression $\{1\ \ 2\}$ of the algebraic object. Enter the list:

$$\{TLIN\ \ \{1\ \ 2\}\}$$

Now press APLSUBEX to get:

$$Y\ \ \left(\frac{-1}{2}\right)\ COS(2\ X) + \frac{1}{2} + \left(Y - \frac{1}{2}\right)\ COS(X)^2$$

What problems does this program have?

Last thing that we do before returning to calculus is visualisation of such algebraic trees using a quite unorthodox method. It would be a cumbersome thing to program a graphics representation of the algebraic tree, but we don't need to do it. The HP49G has a built-in tree graphics generator. It is… the built-in filer!!! If we create an algebraic tree, then we can also create a directory structure based on that tree. For example consider the expression A+B. We can create a directory with the name + (!) and in this directory we can create two sub directories with the names A and B. The filer of the HP49G will display the tree at the right. You may wonder how we can create a directory named +. But this is easy. We can enter the string "+" and used the command S~N to convert it to a name. So let's make a program that takes an algebraic tree and creates the appropriate directory structure:

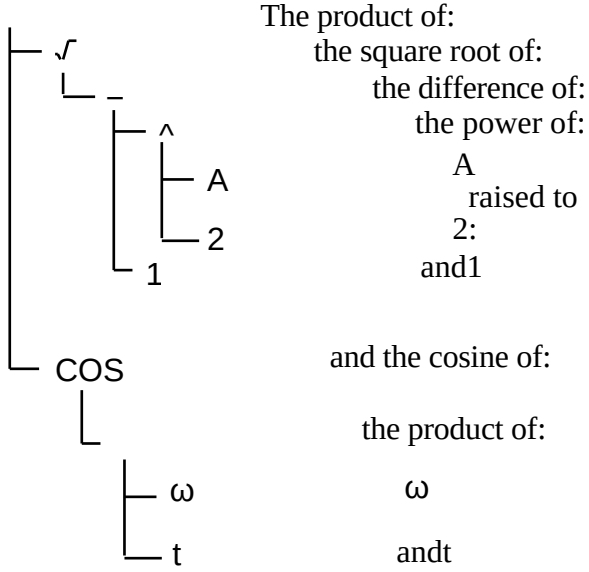

```
<<
  PATH                          @Store current path
  → path
  <<
    1.                          @Do to all elements
    <<                          @of algebraic tree
      CASE
           DUP TYPE 5. ==       @If element is list
        THEN                    @then call yourself
          REVLIST TREE →DSTRUCT
        END
```

```
      DUP TYPE {18. 14.} @If element is command
      SWAP POS           @or function
    THEN
      →STR S~N           @then convert it to name
      WHILE              @Add character "´" to
        VARS OVER POS    @the name until it is
      REPEAT             @unique
        "´" + S~N
      END                @Create sub directory
      DUP CRDIR EVAL     @and switch to that
    END                  @sub directory
    IF                   @If none of the above
      DUP TYPE 6. ≠      @then if element is not a
    THEN                 @name, then convert it
      →STR S~N           @to a name
      WHILE              @Add character "´" to
        VARS OVER POS    @the name until it is
      REPEAT             @unique
        "´" + S~N
      END
    CRDIR                @Create sub directory
  END
 >>
 DOSUBS
 path EVAL               @switch to sub directory
>>                       @we came from
>>
```

This is the program TREE → DSTRUCT. It doesn't display the tree, it only creates the appropriate directory structure. We wrap it in a program that displays the directory structure as a tree.

```
<<
  ALG→TREE               @Create tree list
  TREE→DSTRUCT           @Create directory struct.
  FILER                  @Run filer to view it
  VARS HEAD PGDIR        @Purge directory struct.
>>
```

This is the program VIEWALGTREE. Let's try it. Enter $\sqrt{A^2-1}$ SIN(ω t) and press VIEWALGTREE. After some seconds the HP49G starts the filer and highlights the current directory in the directory tree. Press ▼ a couple of times. You see the tree of the algebraic object in the filer. So that's another possible (mis)usage of the filer as a visualisation tool for algebraic trees. When you have wondered enough about other possible (mis)usage of the filer, press CANCEL. This program will purge the root directory of the tree and all sub directories that were created in it, and then it will exit, The moral of the story is that we can use a screwdriver as a hammer or any other tool, if we only have enough fantasy.



Another consequence of the possible conversion of functions or command to names, is that we can have our own definition of any function. Consider the following example. e. Enter the program:

```
<< → x y
  << IF                    @If we have two numeric
       x TYPE 3. ==        @vectors
       y TYPE 3. == AND    @then find dot product
    THEN x y DOT           @else find product
    ELSE x y *
    END
  >>
>>
```

Enter the string " " and press S~N. You get ' ', which is no more the function , but a name. Press STO. Now you have a new program named ' ', which can be used for dot and normal products. For example enter [1. 2. 3.] and then [4. 5. 6.] (with decimal points after the numbers - we want *numeric* vectors), and press * to get 32. But enter 2 and 3 and press * to get 6. Whenever you press the menu key *, the new extended functionality will be used. But the normal multiplication key × still retains its normal functionality. The same applies when writing programs. When you press * while writing a program, then the multiplication sign which refers to the new program will be placed in the program text. But when you press ×, the sign refers to the built-in multiplication. Now, press ', then *, and then ENTER to put the quoted name ' ' on the stack. Then press PURGE to delete the program named ' '. Just imagine how strongly extendable the command set of the HP49G can be, if we use this capability. And since only the variables are accessible that exist in the current path, we can even have many differently extended functions available in different directories. *This* is extensibility!!!

After this excursion to the forests of algebraic trees, let's return to the rocky paths of calculus. We examine partial derivatives of higher order. Consider a function $f(x,y)$. We can differentiate this more than once. For example we can take the derivative for x twice. Then we write (in mathematics text books):

$$\frac{\partial^2 f(x,y)}{\partial x^2}$$

and on the HP49G:

$$\frac{\partial}{\partial x}\left(\frac{\partial}{\partial x}\left(f(x,y)\right)\right) \text{ or } \mathrm{DERIV}\left(\mathrm{DERIV}\left(f(x,y),x\right),x\right)$$

We can of course also take the "mixed" derivatives, like:

$$\frac{\partial^2 f(x,y)}{\partial x \partial y}$$

for which on the HP49G we write:

$$\frac{\partial}{\partial x}\left(\frac{\partial}{\partial y}\left(f(x,y)\right)\right) \text{ or } \mathrm{DERIV}\left(\mathrm{DERIV}\left(f(x,y),y\right),x\right)$$

Let's have an example. We will find:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial Y}\left(X\ \mathrm{SIN}(Y) - Y\ \mathrm{COS}(X)\right)\right)$$

using RPL syntax.

Enter $X\ \mathrm{SIN}(Y) - Y\ \mathrm{COS}(X)$. Now enter Y and press ∂ to get $X\ \mathrm{COS}(Y) - Y\ \mathrm{COS}(X)$. Enter X and press ∂ again to get $\mathrm{COS}(Y) + \mathrm{SIN}(X)$. We do the same example using algebraic objects. Enter:

$$\frac{\partial}{\partial X}\left(\frac{\partial}{\partial Y}\left(X\ \mathrm{SIN}(Y) - Y\ \mathrm{COS}(X)\right)\right)$$

or

$$\mathrm{DERIV}\left(\mathrm{DERIV}\left(X\ \mathrm{SIN}(Y) - Y\ \mathrm{COS}(X),Y\right),X\right)$$

and press EXPAND to get $\mathrm{SIN}(X) + \mathrm{COS}(Y)$.

The "mixed" derivatives depend on the order of differentiation, i.e. in general:

$$\frac{\partial^2 f(x,y)}{\partial x \partial y} \neq \frac{\partial^2 f(x,y)}{\partial y \partial x}$$

But if the mixed derivatives are continuous functions of x and y in a given domain, then the mixed derivatives are equal to each other in this domain.

You remember that we have done a program for finding the differential of a monovariate function. Now we will extend this concept to functions of more than one variables. The total differential of such a function $z = f(x,y)$ is given by:

$$dz = \frac{\partial f(x,y)}{\partial x}\ dx + \frac{\partial f(x,y)}{\partial y}\ dy$$

The same concept can be extended to functions of even more variables. For example consider $z = f(x_1, x_2, x_3, \ )$. The total differential of this function is:

$$dz = \frac{\partial f(x_1, x_2, x_3, \ )}{\partial x_1}\ dx_1 + \frac{\partial f(x_1, x_2, x_3, \ )}{\partial x_2}\ dx_2 + \frac{\partial f(x_1, x_2, x_3, \ )}{\partial x_3}\ dx_3 +$$

If the partial derivatives are themselves continuous differentiable functions, then we can have total differentials of higher order. For example, consider the function $z = f(x,y)$ again. The total differential of second order is:

$$dz^2 = \frac{\partial\left(\frac{\partial f(x,y)}{\partial x}\ dx + \frac{\partial f(x,y)}{\partial y}\ dy\right)}{\partial x} + \frac{\partial\left(\frac{\partial f(x,y)}{\partial x}\ dx + \frac{\partial f(x,y)}{\partial y}\ dy\right)}{\partial y} =$$

$$\frac{\partial^2 f(x,y)}{\partial x^2}\ dx^2 + 2\ \frac{\partial f(x,y)}{\partial x \partial y}\ dx dy + \frac{\partial^2 f(x,y)}{\partial y^2}\ dy^2$$

The tiny quantities dx, dy, and so on, are considered to be *constant*.

We already made a program for finding the total derivative of a monovariate function. Now we make a program for finding the total derivative of a given order of a function of more variables. The program will take the function from stack level 3, the list of variables from stack level 2, and the order from stack level 1, and will return the total derivative introducing new variables dx, dy, which of course must not appear in the function.

```
<<
  OVER
  1                                  @Make   list   of   dx, dy,
etc.
  <<
    "d" SWAP + S~N
  >>
  DOSUBS
   → f vars ord dvars
  <<
    1 ord
    START
      f vars ∂                       @Differentiate for all vars
      dvars *                        @Multiply each partial
      0 + Σ LIST                     @derivative by the approp.
      EXPAND 'f' STO                 @dvar. Add all expressions
    NEXT
    f
  >>
>>
```

This is the program Td that comes with this document. To try it enter $TAN(X+Y)$, then $\{X\ \ Y\}$, and then 2 (second order), and press Td to get:

$$\left(2\ dX^2 + 4\ dY\ dX + 2\ dY^2\right)\ TAN(X+Y)^3 + \left(2\ dX^2 + 4\ dY\ dX + 2\ dY^2\right)\ TAN(X+Y)$$

If you want to focus on the tiny quantities dx, dy, then enter dx and press REORDER to get:

$$\left(2\ \mathrm{TAN}(X+Y)^3+2\ \mathrm{TAN}(X+Y)\right)\ dX^2+\left(4\ \mathrm{TAN}(X+Y)^3+4\ \mathrm{TAN}(X+Y)\right)\ dY\ dX+\left(2\ \mathrm{TAN}(X+Y)^3+2\ \mathrm{TAN}(X+Y)\right)\ dY^2$$

The meaning of the total differential can be made understandable by a geometric visualisation. It is the total change of a function of more than one variables, when each of the variables changes from x to $x+dx$. The quantity dx is arbitrary small but positive. The total differential is the fundament out of which we can calculate the maximum error of some quantity that is a function of more than one variables. For example, suppose that you are experimenting with an (approximately) ideal gas. You want to calculate its pressure P by measuring its mass m, its temperature T, and its volume V, and use the formula:

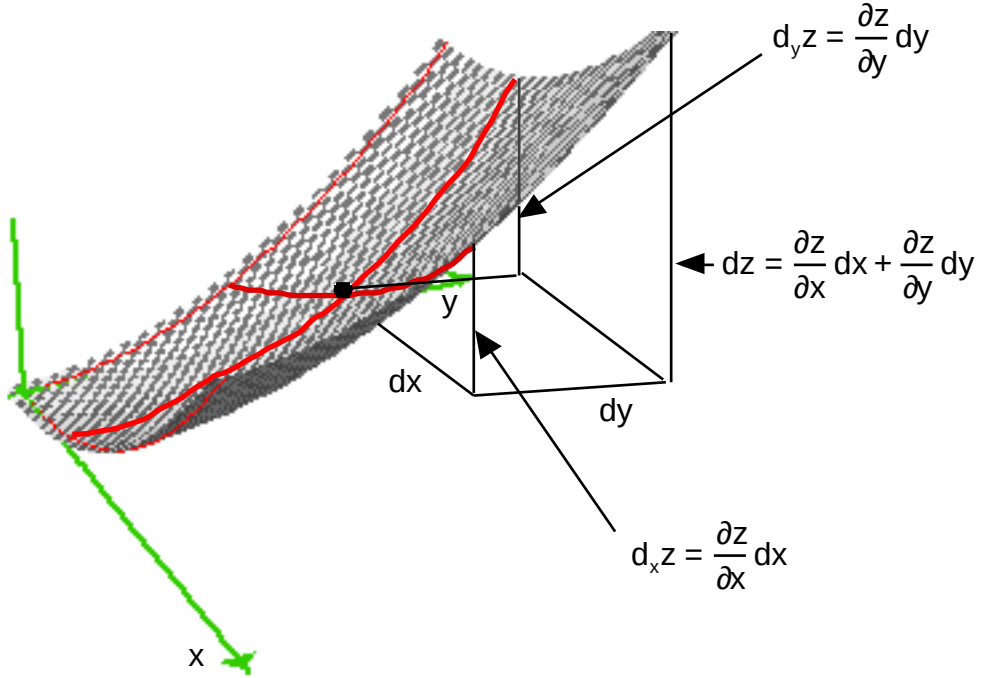
$$P=\frac{m\ R\ T}{M\ V}$$

In this formula the gas constant R, and the molecular weight of the gas M are considered to be known without error. But the measurements of m, T and V can't be perfect. They will be measured with errors Δm, ΔT and ΔV. The total change of the calculated pressure caused by the error in measurements will be:

$$\Delta P=\left|\frac{\partial\frac{m\ R\ T}{M\ V}}{\partial m}\right|\Delta m+\left|\frac{\partial\frac{m\ R\ T}{M\ V}}{\partial T}\right|\Delta T+\left|\frac{\partial\frac{m\ R\ T}{M\ V}}{\partial V}\right|\Delta V=$$

$$\frac{R\ T}{M\ V}\Delta m+\frac{m\ R}{M\ V}\Delta T+\frac{m\ R\ T}{M\ V^2}\Delta V$$

We use the absolute values of the partial derivatives, because they can sometimes be positive and sometimes negative. If we wouldn't use the absolute values, then the error ΔP could be calculated less than its maximum value. We can make a program that derives the expression of the error of a calculated quantity out of the errors of the measured quantities, on which the calculated quantity depends. The program takes as arguments: The calculated quantity as a function of the measured variables from stack level 5, the list of measured



variables from stack level 4, the list of variables that are greater than 0 from stack level 3, the list of variables that are less than 0 from stack level 2, and the list of maximum errors of the measured quantities from stack level 1. It returns the expression $\Delta(\text{variables}) = \text{function}(\text{variables})$ out of which we can create a user defined function by pressing DEF.

```
<<
  PUSH
  { HOME CASDIR REALASSUME } RCL
  → alg vars posvvars negvvars
    maxerrvvars assums
  <<
    -103 CF              @Set real mode
    vars 'Δ' APPLY       @Create expression Δ(vars)
    IF
      posvvars {} ≠      @If we have positive vars.
    THEN
      posvvars 1         @make appropriate assumptions
      <<                 @for calculating the abs. value
        "≥0" + "'" SWAP +
        "'" + OBJ→ ASSUME
        DROP
      >>
      DOSUBS
    END
    IF
      negvvars {} ≠      @If we have negative vars.
    THEN
      negvvars 1         @make appropriate assumptions
      <<                 @for calculating the abs. value
        "≤0" + "'" SWAP +
        "'" + OBJ→ ASSUME
        DROP
      >>
      DOSUBS
    END
    alg vars ∂ ABS       @Find maximum error
    EXPAND
    vars 1
    <<                   @Create Δvars
      "Δ" SWAP + S~N
    >>
    DOSUBS
    DUP UNROT * 0 +
    Σ LIST
    SWAP maxerrvvars 2   @Create equations
    << = >>              @Δvar=value
    DOLIST
    1                    @and substitute in
    << SUBST >>          @expression
    DOSUBS =
    { HOME CASDIR } EVAL @Restore original
    assums 'REALASSUME' STO @assumptions
  >>
  POP
>>
```

This is the program ΔMSRM. We test it with the above example of the ideal gas. Enter:

$$\frac{m\ R\ T}{M\ V}$$

then the list of measured quantities $\{m\ \ T\ \ V\}$, the list of positive quantities $\{m\ \ T\ \ V\ \ M\ \ R\}$, the list of negative quantities $\{\ \}$, and the list of maximum errors in the measured quantities $\{.00001\ \ .01\ \ .001\}$. (We could also enter $\{\Delta m\ \ \Delta T\ \ \Delta V\}$ if we want to have the symbolic result.) Press ΔMSRM to get:

$$\Delta(m,T,V) = \frac{T\ R}{V\ M}\ .00001 + \frac{m\ R}{V\ M}\ .01 + \frac{m\ T\ R}{V^2\ M}\ .001$$

If you now press DEF you will have the user function Δ which will take the measured values of m, T and V from the stack, and return the maximum error for P, under the specified maximum errors of the measured quantities. (Of course you must substitute a numeric value for

the gas constant R in the formula.)

Since we have examine derivatives of functions of more than one variables, it is time to take a look at further possible usage of formal derivatives of more than one variables. For the derivative $\frac{\partial f(x,y)}{\partial x}$ the HP49G writes d1f(x,y), when the function f is undefined. Similarly, for the derivative $\frac{\partial f(x,y)}{\partial y}$ it writes d2f(x,y), and for the derivative $\frac{\partial^2 f(x,y)}{\partial x \partial y}$ it writes d1d2f(x,y). We make an example of the usage of such formal derivatives as user defined functions.

We create first a sub directory. We are going to create some variables and so it is a good policy to create them in a separate directory for clarity and for preventing cluttering of the menu VAR. Enter something like 'EXMP1' or similar and press CRDIR. Switch to directory 'EXMP1'. Here we will do the whole work for the first example.

In the first example we want to find a formula for the difference $C_P - C_V$ between the heat capacity of a system at constant pressure and its heat capacity at constant volume. First the theory.

From general thermodynamics we have the definition of heat capacity C_v of a system at constant volume:

$$C_v = \left(\frac{\partial U}{\partial T}\right)_V \qquad (1)$$

U is the inner energy of the system, a function of its absolute temperature T and of its volume V. The differential quotient $\left(\frac{\partial U}{\partial T}\right)_V$ in parentheses with the index V, denotes that we take the partial derivative for T by holding the volume V constant.

Similarly the definition of heat capacity C_P of a system at constant pressure is:

$$C_P = \left(\frac{\partial H}{\partial T}\right)_P \qquad (2)$$

H is the enthalpy of the system, a function of its absolute temperature T and of the pressure P. The differential quotient $\left(\frac{\partial H}{\partial T}\right)_P$ in parentheses with the index P, denotes that we take the partial derivative for T by holding the pressure P constant.

The definition of the enthalpy H is:

$$H = U + P\ V \qquad (3)$$

Substituting this in (2) we get:

$$C_P = \left(\frac{\partial H}{\partial T}\right)_P = \left(\frac{\partial (U + P\ V)}{\partial T}\right)_P = \left(\frac{\partial U}{\partial T}\right)_P + \left(\frac{\partial P}{\partial T}\right)_P V + P \left(\frac{\partial V}{\partial T}\right)_P$$

Now, since in this derivation we hold the pressure P constant, the term

$$\left(\frac{\partial P}{\partial T}\right)_P V$$

is 0. So we get:

$$C_P = \left(\frac{\partial H}{\partial T}\right)_P = \left(\frac{\partial U}{\partial T}\right)_P + P \left(\frac{\partial V}{\partial T}\right)_P$$

We build up the difference $C_P - C_V$:

$$C_P - C_V = \left(\frac{\partial U}{\partial T}\right)_P + P\left(\frac{\partial V}{\partial T}\right)_P - \left(\frac{\partial U}{\partial T}\right)_V$$

Now let's do that on the HP49G. Enter the definition of C_P:

$$Cp = \frac{\partial}{\partial T}(H(T))$$

and press DEF, which stores the expression $\frac{\partial}{\partial T}(H(T))$ in variable Cp. Enter the definition of C_v:

$$Cv = \frac{\partial}{\partial T}(U1(T))$$

and press DEF again to store $\frac{\partial}{\partial T}(U1(T))$ in Cv. We use $U1(T)$ to distinguish between the inner energy as a function of the temperature T and the volume V from the inner energy $U2$ as a function of the pressure P and the temperature T. Enter the definition of the enthalpy:

$$H(T) = U2(T) + P\ V(T)$$

and press DEF to create the user defined function H.

Enter $Cp - Cv$ and press EXPAND to get the result $-(d1U1(T) - (d1U2(T) + P\ d1V(T)))$, which is the same as:

$$\left(\frac{\partial U}{\partial T}\right)_P + P\left(\frac{\partial V}{\partial T}\right)_P - \left(\frac{\partial U}{\partial T}\right)_V$$

but written differently. If you want you can use now our program $dn \rightarrow dv$ to beautify the result and get:

$$-\left(\frac{\partial}{\partial T}(U1(T)) - \frac{\partial}{\partial T}(U2(T)) + P\ \frac{\partial}{\partial T}(V(T))\right)$$

For an ideal gas we have:

$$\left(\frac{\partial V}{\partial T}\right)_P = \alpha\ V$$

where α is the expansivity of the ideal gas. Enter $d1V(T) = \alpha\ V(T)$ and press DEF to create the user defined derivative $d1V$. Let's find the difference $C_P - C_V$ for an ideal gas. Enter again $Cp - Cv$ and expand to get $-(d1U1(T) - (d1U2(T) + P\ \alpha\ V(T)))$. Again you can beautify this with $dn \rightarrow dv$ to get:

$$-\left(\frac{\partial}{\partial T}(U1(T)) - \frac{\partial}{\partial T}(U2(T)) + P\ \alpha V(T)\right)$$

Now enter CLVAR to purge all variables in the directory. We will do the same example using different variable definitions. Enter $Cp = d1H(T)$ and press DEF. Then enter $Cv = d1U1(T)$ and press DEF again. We have defined the heat capacities using dn syntax. We will work with an ideal gas again, so enter $d1V(T) = \alpha\ V(T)$ and press DEF to create the user defined derivative $d1V$. Now instead of the user function H we are going to use the user defined derivative $d1H$. Enter $d1H(T) = d1U2(T) + P\ d1V(T)$. Press DEF to make the definition. Enter $Cp - Cv$ and expand to get $-(d1U1(T) - (d1U2(T) + P\ \alpha\ V(T)))$ again.

Yet another way to so that. Enter CLVAR to clean up the directory. Now we will use the definition of the inner energy as a user function.

Enter $U2(T) = H(T) - P\ V(T)$ and press DEF to create the user function $U2$. We will use the derivative:

$$\frac{\partial}{\partial T}(U1(T)) = Cv$$

Enter $d1U1(T) = Cv$ and press DEF. This creates the user defined derivative $d1U1(T)$.

We will also use the derivative of the enthalpy:

$$\frac{\partial}{\partial T}(H(T)) = Cp$$

Enter $d1H(T) = Cp$ and press DEF to create the definition of the user defined derivative.

Now, enter:

$$\frac{\partial}{\partial T}(U2(T)) - \frac{\partial}{\partial T}(U1(T))$$

and expand to get $-(Cp - (Cv - P\ d1V(T)))$, which beautified would be:

$$-\left(Cp - \left(Cv - P\ \frac{\partial}{\partial T}(V(T))\right)\right)$$

Enter:

$$\frac{\partial}{\partial T}(U2(T)) - \frac{\partial}{\partial T}(U1(T)) + P\ \frac{\partial}{\partial T}(V(T))$$

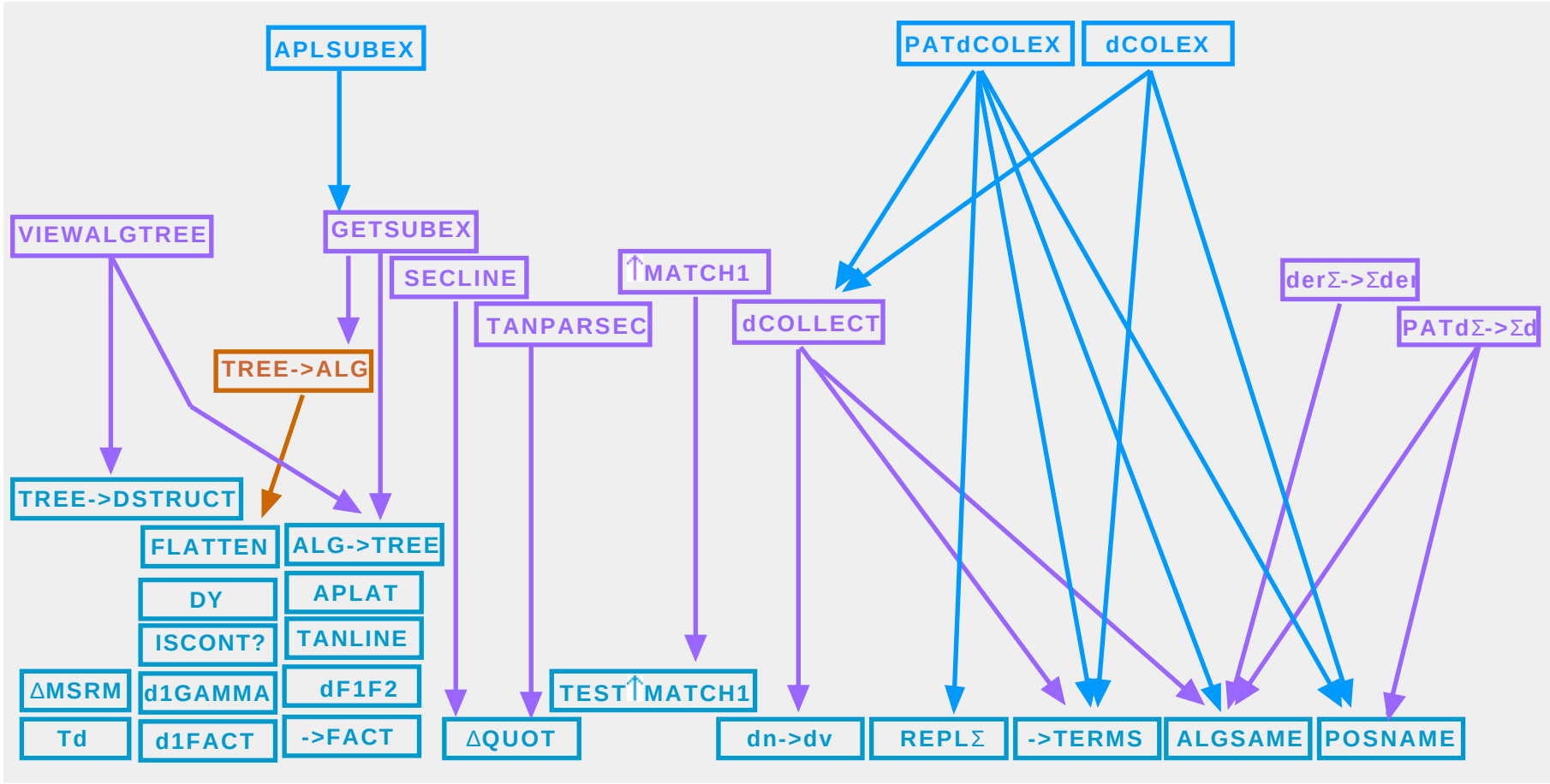
and expand again to get $-(Cp - Cv)$.

The above examples show how well the HP49G can handle such formal derivatives. Especially the third demonstrates how to get results with "normal" variables out of only formally defined derivatives.

We are at the end of the first volume of the Basic Calculus Marathon. In the second volume we will continue with extrema of functions and other interesting things.

Before we go to sleep with smoking heads dreaming of the derivative of the derivative of Ouzo(Trabakoulas), we take a look at our program building (next page), which gets more and more crowded. How am I going to represent it when even more programs come?

$$\frac{\partial}{\partial Nick}(Greetings(Nick))$$


APLSUBEX
PATdCOLEX
dCOLEX
VIEWALGTREE
GETSUBEX
SECLINE
↑MATCH1
derΣ->Σder
TANPARSEC
dCOLLECT
PATdΣ->Σd
TREE->ALG
TREE->DSTRUCT
FLATTEN
ALG->TREE
DY
APLAT
ISCONT?
TANLINE
ΔMSRM
d1GAMMA
dF1F2
TEST↑MATCH1
Td
d1FACT
->FACT
ΔQUOT
dn->dv
REPLΣ
->TERMS
ALGSAME
POSNAME