

Maple Procedures for Partial and Functional Derivatives

E.S.Cheb-Terrab *

Instituto de Física
Departamento de Física Teórica
Universidade do Estado do Rio de Janeiro
RJ 20550 - Brazil

Abstract

A package with new **Maple V** commands for the evaluation of partial derivatives, functional derivatives and integrals containing the Dirac delta function is presented. Some examples of the use of these commands in physics are shown.

*Computer Physics Communications, 79, E. S. Cheb-Terrab, "Maple procedures for partial and functional derivatives", 409-424 (1994), with kind permission from Elsevier Science B.V., Amsterdam, The Netherlands

Introduction

As is well known, partial and functional derivatives play a major role in calculations in physics. On the other hand, even though the advent of symbolic computing has made it possible to automate these operations, functional derivatives are not yet implemented in general purpose systems, and the evaluation of partial derivatives requires special care[1, 2], restricting the use of these programs in research.

The subject of this paper is related to these topics, and consists of the presentation of a set of new **Maple V** differentiation commands¹ which both overcome some of the present restrictions (with respect to the evaluation of partial derivatives) and allow functional derivatives to be calculated.

The plan for the exposition is as follows. In Sec.1, the whole package of new commands is presented through a short *write-up*, followed by a long *write-up* for the most relevant routines, mainly **pdiff** for partial derivatives and **fdiff** for functional derivatives². This long *write-up* also contains examples which can be viewed as simple *input/output* tests for the routines of the package. Sec.2 was written as an *input/output* test in the context of “problems in physics”, aiming at giving a more concrete illustration of the new commands’ abilities as well as an idea of the type of problems which can be tackled with them. Finally, the conclusion contains a short comment about possible improvements, in order to extend the use of general purpose symbolic algebra systems in theoretical physics.

Aside from this, some additional information is given in two appendices. The first one contains a brief discussion about the use of *mappings* in the calculation of partial derivatives, and the second one explains some new **type/...** subroutines, as well as a slight change made to the **D/procedure** standard routine, in order to make it compatible with the package here presented.

1 The *partials* package

1.1 Short *write-up*

A brief review of the commands of the package is as follows:

- **pdiff** performs the calculation of partial derivatives of algebraic expressions with respect to parameters or functions. It does not require the previous definition of a mapping. It also allows the User to freeze specified functions of the derivation variables, in order to arrive at “explicit” (fixed) partial derivatives, commonly used in physics.

¹The commands and routines here presented only work with the **Maple V** Release 5.2 environment and are not intended to work with any other computer algebra system.

²Aside from this, the package itself contains an *ON-LINE* help in standard **Maple** format which can be viewed as a User-manual for all the routines.

- **usepdiff** takes full advantage of expressions involving the **D** standard operator for the evaluation of partial derivatives by sending the task to **pdiff**, allowing, in this way, the calculation to be done without the previous building of a mapping.
- **fdiff** calculates functional derivatives of algebraic expressions (containing or not multiple integrals).
- **evalDi** evaluates integrals containing the Dirac delta function or its derivatives.
- **parameters** allows the parameters of a theory to be defined in such a way that no functionality can be attached to them during the **Maple** session. In this way, a parameter defined through this command will behave not as a variable (standard behavior), but rather as if it were a “number” (fixed parameter).
- **odiff** returns the general differential order (or particular, with respect to an indicated variable) of an expression containing partial derivatives.
- **seldiff** selects, from an algebraic expression, those terms which contain derivatives of an indicated order with respect to an indicated variable.
- **usediff** and **useD** are syntax converters between the **diff** and the **D** standard **Maple** syntaxes for derivatives.

They are equivalent to the **convert(...,diff)** and **convert(...,D)** standard **Maple** routines. The main differences are: **usediff** does not return results through the **&where** function as does **convert(...,diff)**, and **useD** realizes the conversions to the **D** syntax even inside composite functions, while **convert(...,D)** does not.

- **Intc**, **Int2c**, **Int3c**, and **Int4c** are simple macros that shorten the input of simple, double, triple and quadruple integrals from $-\infty$ to $+\infty$.
- A new **Value** routine, with a definition slightly different from the standard value routine, allows the User to define Value/... rules for functions with indexed names (specially useful when working with Relativistic theories).

1.2 Long *write-up*

The **Maple** standard differentiation commands are **diff**, **D** and **Diff**. Because they already conform a powerful set, the goal of the *partials* package was not to “substitute” these commands but to create new ones (mainly **pdiff**, **fdiff**), which work with the standard ones in a complementary manner, extending the environment’s ability in that:

1. they permit the User to calculate partial derivatives, explicit (fixed) or general ones, of algebraic expressions, with respect to parameters or functions, without defining mappings;
2. they allow functional derivatives to be calculated.

A detailed description of **pdiff** and **fdiff**, along with some examples which can be viewed as simple tests for the *input/output* of the commands, is as follows:

Command name: **pdiff**

Feature: Partial (general or explicit) differentiation command

Calling sequence:

```
> pdiff(a, x1, x2, ..., xn);  
> pdiff(a, x1, x2, ..., xn, {f1(x,...), f2(x...), ...});
```

Parameters:

a	- an algebraic expression (the “derivand”)
x1, ...	- names or functions (the derivation variable(s))
{f1(x,...), ...}	- optional; indicates which functions (and their derivatives) of the derivation variables must be frozen

Synopsis:

This function is part of the *partials* package, and so can be used in the form **pdiff(..)** only after performing the instruction

```
> with(partials): (or)  
> with(partials, pdiff):  
at the Maple prompt “>”.
```

pdiff calculates the partial derivative, general or “explicit”, of algebraic expressions with respect to parameters or functions (the **diff** command cannot differentiate with respect to functions), without requiring the definition of a mapping for the “derivand” (required when using the **D** operator).

In order to calculate “explicit partial derivatives” (also not possible with the **diff** command), **pdiff** requires the indication of which functions of the derivation variables must be “frozen” (fixed) during the calculation. This situation typically happens when using variational principles which, in turn, introduce functions as parameters³. If a function is indicated inside the braces, all its derivatives will also be frozen.

It makes no difference whether to use the **diff**, **Diff**, or **D** command in the arguments passed to **pdiff**, since it will take into account the mathematical equivalence between them. For example, instructions such as **pdiff(L, diff(q(t), t))**, **pdiff(L, Diff(q(t), t))** and **pdiff(L, D(q)(t))** will all give the same result.

As a rule, **pdiff** will try not to change the User’s choice of syntax (**diff**, **Diff**, or **D**). Nevertheless, when the arguments sent to **pdiff** make use of more than one syntax to represent the same mathematical object, the syntax selected by **pdiff** for the output will be that of the “derivand”. If the derivand uses more than one syntax representing the same mathematical object, the result will be returned using the **D** syntax for that object.

If the algebraic expression is given with the functional dependence indicated but in implicit form (for example: $f(x, y, z)$, with unknown form for f), the result will be returned

³This is the case, for example, of the Lagrange formalism with the related Lagrange equations in mechanics.

using the **D** operator. In this case, the returned expression can be transformed into a concrete result through the **usepdiff** command, after defining the explicit form of the functions involved by directly assigning values to them (mappings are not mandatory).

Special rules for the evaluation of partial derivatives with respect to User Defined functions using **pdiff** can be stated by indicating the rule (for the NAME of the U.D. function) to the **diff** command in **Maple**'s usual way.

*Examples*⁴:

```
> with(partials,pdiff);
```

[pdiff]

```
> pdiff(cos(q(t)),q(t));
```

$$-\sin(q(t)) \tag{1}$$

```
> Int(f(x)*exp(y(t)),x=0..y(t)^(1/2));
```

$$\int_0^{\sqrt{y(t)}} f(x)e^{y(t)} dx$$

```
> pdiff(",y(t));
```

$$e^{y(t)} \int_0^{\sqrt{y(t)}} f(x)dx + \frac{f(\sqrt{y(t)})e^{y(t)}}{2\sqrt{y(t)}} \tag{2}$$

```
> alias(Q=(q(t),diff(q(t),t),t)): # defines a "Q" functionality.
```

```
> pdiff(L(Q),t); # The total derivative:
```

$$D_{[1]}(L)(Q)\frac{\partial}{\partial t}q(t) + D_2(L)(Q)\frac{\partial^2}{\partial t^2}q(t) + D_{[3]}(L)(Q) \tag{3}$$

```
> L(Q) := 1/2*diff(q(t),t)^2-1/2*k * q(t)^2 - cos(w*t); # Assigns a
# value to L(Q)
```

$$L(Q) := \frac{1}{2}\frac{\partial}{\partial t}q(t)^2 - \frac{1}{2}kq(t)^2 - \cos(wt) \tag{4}$$

```
> usepdiff(""); # The total derivative eq(3) without mappings:
```

$$-kq(t)\frac{\partial}{\partial t}q(t) + \left(\frac{\partial}{\partial t}q(t)\right)\frac{\partial^2}{\partial t^2}q(t) + \sin(wt)w \tag{5}$$

⁴In what follows, the *input* can be recognized by the **Maple** prompt **>**, and the output was obtained with **Maple** release 5.2. *for Windows*, in a DOS platform. Further, special care was taken to keep the *input* and *output* shown along this paper with almost exactly the same format and aspect as that which appears on the computer's screen.

```
> pdiff(L(Q),q(t));      # Each partial derivative:
```

$$-kq(t) \tag{6}$$

```
> pdiff(L(Q),D(q)(t));
```

$$\frac{\partial}{\partial t}q(t) \tag{7}$$

```
> pdiff(L(Q),t,{q(t)}); # the "explicit" partial derivative:
```

$$\sin(\omega t)\omega \tag{8}$$

Command name: **fdiff**

Feature: Evaluation of functional derivatives

Calling sequence:

```
> fdiff(a, f1);
> fdiff(a, f1, f2, ..., fn);
```

Parameters:

a - an algebraic expression
f1, ... - functions (the functional differentiation variables)

Synopsis:

This function is part of the *partials* package, and so can be used in the form **fdiff(...)** only after performing the instruction

```
> with(partials): (or)
> with(partials,fdiff):
```

at the **Maple** prompt.

fdiff calculates functional derivatives of algebraic expressions.

If the expression being differentiated contains an integral (maybe a multiple or nested one), **fdiff** goes inside it and, after the functional differentiation of the integrand, tries to evaluate the resulting integral which now contains the Dirac delta function or its derivatives. This evaluation, in turn, will be carried out only if the integration limits are from $-\infty$ to $+\infty$ and the argument of the Dirac delta function involved is a linear function of the integration variable. Otherwise, the *intermediate* integral without evaluation will be returned (after replacing all occurrence of **int** by its inert form **Int**). In this way, it is possible to control the different steps of a calculation through a simple change of the integration limits. The evaluation of these integrals is performed by the **evalDi** command.

Since **fdiff** is implemented using **pdiff**, which, in turn, makes use of **diff**, the differentiation knowledge of **fdiff** with respect to mathematical functions is exactly that of **diff**, so any special rule for the functional derivative of a User Defined function can be stated by indicating the rule (for the NAME of the U.D. function) to **diff** in **Maple**'s usual way. In

addition, since **pdiff** recognizes the derivatives built with the **diff**, **Diff**, or **D** syntaxes as mathematically equivalent, the same holds for **fdiff**.

*Examples*⁵:

```
> alias(delta=Dirac):
> with(partials,fdiff);
```

[fdiff]

```
> fdiff(q(u1,u2,u3,u0),q(x1,x2,x3,x0));
```

$$\delta(u1 - x1) \delta(u2 - x2) \delta(u3 - x3) \delta(u0 - x0) \quad (9)$$

```
> diff(q(u,v),u,v$2);
```

$$\frac{\partial^3}{\partial u \partial v^2} q(u, v)$$

```
> fdiff(",q(x,y));
```

$$\delta(1, u - x) \delta(2, v - y) \quad (10)$$

```
> Intc(1/2*diff(q(t),t)^2-1/2*k*q(t)^2,t);
```

$$\int_{-\infty}^{\infty} \frac{1}{2} \frac{\partial}{\partial t} q(t)^2 - \frac{1}{2} k q(t)^2 dt$$

```
> fdiff(",q(u));
```

$$-kq(u) - \frac{\partial}{\partial u} D(q)(u) \quad (11)$$

```
> subsop(2=t,""); # Changing the integration limits in eq("")
```

$$\int \frac{1}{2} \frac{\partial}{\partial t} q(t)^2 - \frac{1}{2} k q(t)^2 dt$$

```
> fdiff(",q(u)); # The intermediate integrals are returned
```

$$-k \int \delta(t - u) q(t) dt + \int D(q)(t) \delta(1, t - u) dt \quad (12)$$

⁵Maple's convention for the derivative of the Dirac delta function is given by:

$$\frac{\partial^n}{\partial x^n} \delta(x) = \delta(n, x)$$

2 Test for the *input/output* of the package

What follows is an *input/output* test for the *partials* package, in the context of “problems in physics”⁶. This context is in order to illustrate the new commands’ abilities, while giving the User some insight on the kind of problems in which these commands can be used.

2.1 Test 1: Poisson brackets in field theories

As the first test, let’s consider the building of the matrix of the Poisson brackets of constraints in the canonical quantization of Maxwell’s theory in four dimensions. The expected result is given in reference[3]. First, the main commands and some alias are introduced:

```
> with(partials,fdiff,evalDi,Intc);
```

$$[Intc, evalDi, fdiff]$$

```
> with(linalg,matrix):
```

```
> alias(delta=Dirac): # The string delta will represent Dirac.
```

```
> alias(x=(x1,x2,x3,x0)):
```

```
> alias(y=(y1,y2,y3,y0)):
```

```
> alias(z=(z1,z2,z3,z0)):
```

```
> alias(A=(A1,A2,A3,A0)):
```

```
> alias(P=(P1,P2,P3,P0)):
```

(13)

These alias allow a short *input/output* for the space-time parameters (here represented by x , y or z), for the vector field A^μ , and for its canonical momentum P^μ . Next, the Poisson brackets,

$$\{F(x), G(y)\}_{[x_0=y_0]} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \sum_{k=1}^4 \frac{\delta F}{\delta P_{[k]}(z)} \frac{\delta G}{\delta A_{[k]}(z)} - \frac{\delta G}{\delta P_{[k]}(z)} \frac{\delta F}{\delta A_{[k]}(z)} dz_1 dz_2 dz_3 \quad (14)$$

can be entered as:

```
> PB := (F,G) -> evalDi(Int3c(sum('fdiff'(F,P[k](z))*'fdiff'(G,A[k](z))
> -'fdiff'(G,P[k](z))*'fdiff'(F,A[k](z)),
> k=1..4), z1, z2, z3):
```

The **evalDi** command is required for the evaluation of the integrals containing the Dirac delta function, which will appear as a result of the calculation of functional derivatives performed by **fdiff**. The evaluation of the Poisson brackets using this expression will test the *output* of **evalDi** and **fdiff**, as well as that of **pdiff**, **odiff**, **useD**, and **usediff** (all these commands will be called by **fdiff**), at once.

Now, in order to evaluate these brackets at “equal time”, one may declare that the corresponding Dirac delta functions are equal to one⁷:

⁶For a simple *input/output* sequence of partial and functional derivatives without context see the *write-up* in Sec.1.

⁷This would be incorrect in other contexts, but is justified here because it is a very economical way of implementing the “equal time” request of the calculation of the Poisson brackets.

```
> delta(x0-z0) := 1: delta(y0-z0) := 1: delta(x0-y0) := 1:
```

(15)

To arrive at the desired matrix, the constraints of the model are entered as[3]:

```
> c[1] := P0:
> c[2] := D[1](P1)+D[2](P2)+D[3](P3):
> c[3] := A0:
> c[4] := D[1](A1)+D[2](A2)+D[3](A3):
```

(16)

where $c[1]$, $c[2]$, $c[3]$, and $c[4]$ represent the constraints $P_0(\approx 0)$, $\nabla\vec{P}(\approx 0)$, $A_0(\approx 0)$, and $\nabla\vec{A}(\approx 0)$, respectively. The space-time dependence is not yet indicated, in order to allow different space-time parameters to be used when calculating the functional derivatives inside the Poisson brackets. Finally, the matrix can be built as:

```
> C := (i,j) -> PB(c[i](x),c[j](y)):
> matrix(4,4,C);
```

$$\begin{aligned}
& [0, 0, \delta(x1 - y1)\delta(x2 - y2)\delta(x3 - y3), 0] \\
& [0, 0, 0, \delta(2, y1 - x1)\delta(x2 - y2)\delta(x3 - y3) \\
& + \delta(x1 - y1)\delta(2, y2 - x2)\delta(x3 - y3) + \delta(x1 - y1)\delta(x2 - y2)\delta(2, y3 - x3)] \\
& [-\delta(x1 - y1)\delta(x2 - y2)\delta(x3 - y3), 0, 0, 0] \\
& [0, -\delta(2, y1 - x1)\delta(x2 - y2)\delta(x3 - y3) - \delta(x1 - y1)(\delta(2, y2 - x2))\delta(x3 - y3) \\
& - \delta(x1 - y1)\delta(x2 - y2)\delta(2, y3 - x3), 0, 0]
\end{aligned}$$

(17)

This result includes the Laplacian of a tri-dimensional Dirac delta function and checks *OK* with that of [3].

2.2 Test 2: Lagrange equations in field theories

The second test is related to the problem of the equivalence between two Lagrange functions (a *standard* one and its *first order* equivalent⁸) in the context of a field theory with *high order derivatives*. This equivalence has already been derived by hand for the specific case considered below[4], and its derivation will be done here using the new commands.

To start with, let's introduce the **fdiff** command, **Value/...** rules for the operators ∂_μ and ∂^μ (here represented by $d_{-[\mu]}$ and $d_{[\mu]}$ respectively), and for the Laplace (Δ) and d'Alembert (\square) differential operators, as follows⁹:

⁸A *first order* Lagrange function is linear in the derivatives with respect to time. This linearization is obtained by departing from the *standard* Lagrange function and then extending the configuration space through the introduction of auxiliary fields.

⁹The \square character in the *input* lines can be obtained by pressing the combination of keys **Ctrl+Backspace** at the input line of the **Maple for Windows** worksheet. In platforms where this character is not allowed one can work with any other valid string.

```
> with(partials,fdiff,Value);
```

[fdiff, Value]

```
> 'Value/d_':= proc(p) option 'i n d e x e d';
>     diff(op(p),cat(x,op(op(0,p))))
>     end:
>
```

```
>'Value/d':= proc(p) option 'i n d e x e d';
>     if member(op(op(0,p)),{1,2,3}) then -1 else 1 fi:
>     "*diff(op(p),cat(x,op(op(0,p))))
>     end:
>
```

```
> 'value/Delta' := u ->
>     diff(op(u),x1,x1) + diff(op(u),x2,x2) + diff(op(u),x3,x3);
```

$$value/Delta := u \mapsto \frac{\partial^2}{\partial x_1^2}u + \frac{\partial^2}{\partial x_2^2}u + \frac{\partial^2}{\partial x_3^2}u \quad (18)$$

```
> 'value/□' := u -> diff(op(u),x0,x0)
>     - diff(op(u),x1,x1) - diff(op(u),x2,x2) - diff(op(u),x3,x3);
```

$$value/\square := u \mapsto -\frac{\partial^2}{\partial x_1^2}u - \frac{\partial^2}{\partial x_2^2}u - \frac{\partial^2}{\partial x_3^2}u + \frac{\partial^2}{\partial x_0^2}u \quad (19)$$

Since ∂_μ and ∂^μ are being defined as indexed functions, both required the definition of their evaluation rules through the new **Value** command.

Next, some aliases (x and u) for the representation of the space-time parameters, and $(\phi, \pi_{[1]}, \pi_{[2]}, \pi_{[3]})$ for the main (ϕ) and auxiliary fields $(\pi_{[...]})$ as functions of x :

```
> alias(x=(x1,x2,x3,x0)):
> alias(u=(u1,u2,u3,u0)):
> alias(phi = phi(x1,x2,x3,x0)):
> alias(pi[1] = pi[1](x1,x2,x3,x0)):
> alias(pi[2] = pi[2](x1,x2,x3,x0)):
> alias(pi[3] = pi[3](x1,x2,x3,x0));
```

[I, x, u, φ, π_[1], π_[2], π_[3]] (20)

Now the standard (L) and first order ($L1$) Lagrange functions are entered as:

```
> L := 1/2 * Sum(d_[i](phi)*d[i](phi),i = 0..3)
>     - m/2*phi^2 + 1/kappa^2*'□'(phi)^2 + lambda*phi^4/12;
```

$$L := \frac{1}{2} \sum_{i=0}^3 \left(d_{[i]}(\phi) d_{[i]}(\phi) \right) - \frac{m\phi^2}{2} + \frac{\lambda\phi^4}{12} + \frac{(\square(\phi))^2}{\kappa^2} \quad (21)$$

(note the presence of high order derivatives in the $(\square(\phi))^2$ term)

```

> L1 := Sum(d_[i](phi)*d[i](phi),i = 1..3)/2-m*phi^2/2+lambda*phi^4/12
> -pi[1]^2/2+2/kappa^2*pi[1]*pi[3]-pi[2]^2/kappa^2
> -2/kappa^2*pi[1]*diff(pi[2],x0)-2/kappa^2*pi[3]*diff(phi,x0)
> +pi[1]*diff(phi,x0)+4/kappa^2*Delta(pi[1])*diff(phi,x0)
> -2/kappa^2*Delta(pi[1])*pi[1]+Delta(phi)^2/kappa^2;

```

$$\begin{aligned}
L1 := & \frac{1}{2} \sum_{i=1}^3 \left(d_{[i]}(\phi) d_{[i]}(\phi) \right) - \frac{m\phi^2}{2} + \frac{\lambda\phi^4}{12} \\
& - \frac{1}{2} \pi_{[1]}^2 + \frac{2\pi_{[1]}\pi_{[3]}}{\kappa^2} - \frac{\pi_{[2]}^2}{\kappa^2} - \frac{2\pi_{[1]}}{\kappa^2} \frac{\partial}{\partial x0} \pi_{[2]} - \frac{2\pi_{[3]}}{\kappa^2} \frac{\partial}{\partial x0} \phi + \pi_{[1]} \frac{\partial}{\partial x0} \phi \\
& + \frac{4\Delta(\pi_{[1]})}{\kappa^2} \frac{\partial}{\partial x0} \phi - \frac{2\Delta(\pi_{[1]})\pi_{[1]}}{\kappa^2} + \frac{(\Delta(\phi))^2}{\kappa^2}
\end{aligned} \tag{22}$$

where m represents the mass of the particles associated to the ϕ field, λ is a parameter of the theory and $L1$ contains only first order derivatives with respect to $x0$.

L and $L1$ are equivalent if they lead to the same equation for the ϕ field, after the elimination of all the auxiliary fields $\pi_{[...]}$, through their own Lagrange equations. Therefore, to prove this equivalence, the calculation of all the field equations is required. These equations will be obtained by taking the functional derivative of the action S with respect to each field, using the new commands.

The procedure of calculation of the field equations (here named LE) is now built as the functional derivative of a given action S with respect to a given field Q :

```

> LE := (S,Q) -> usediff(subs({u0=x0,u1=x1,u2=x2,u3=x3},fdiff(S,Q(u)))):
\tag{23}

```

the main part of the calculation being done by $fdiff(S,Q(u))$, while the other parts of the procedure aim at a readable output. The action for L and the movement equation for ϕ (here named EQ) are given by:

```

> S := Int4c('Value(L)',x1,x2,x3,x0);
\tag{24}

```

$$S := \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} Value(L) dx1 dx2 dx3 dx0$$

```

> EQ := LE(S,op(0,phi)):
\tag{25}

```

In the same way, the action for $L1$ and the Lagrange equations (here named $EQ1_{[...]}$) for ϕ and for the auxiliary fields, obtained from this action, are calculated through:

```

> S := Int4c('Value(L1)',x1,x2,x3,x0);

```

$$S := \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \text{Value}(L1) dx1 dx2 dx3 dx0 \quad (26)$$

```
> for j in [phi, pi[1], pi[2], pi[3]] do EQ1[j] := LE(S, op(0, j)) od:
(27)
```

The next step is connected with the elimination of the auxiliary fields using the equations above. This elimination does not use the commands of the package but is necessary to conclude the test. To proceed, let's first check the number of terms and the functional dependence of each movement equation¹⁰:

```
> map(nops, EQ1);
table([phi = 16, pi[1] = 10, pi[2] = 2, pi[3] = 2])
(28)
```

```
> map(indets, EQ1, Function);
table([phi = {phi, pi[1], pi[3]}, pi[1] = {phi, pi[1], pi[2], pi[3]}, pi[2] = {pi[1], pi[2]}, pi[3] = {phi, pi[1]}])
(29)
```

Using the information above, one can eliminate the auxiliary fields from all the field equations at once, taking into account each of the equations for the auxiliary fields at a time, through¹¹:

```
> EQ1 := map(u -> simplify(u, {EQ1[pi[3]]}, {pi[1]}), EQ1):
> EQ1 := map(u -> simplify(u, {EQ1[pi[2]]}, {pi[2]}), EQ1):
> EQ1 := map(u -> simplify(u, {EQ1[pi[1]]}, {pi[3]}), EQ1):
> map(indets, EQ1, Function);
table([phi = {phi}, pi[1] = {}, pi[2] = {}, pi[3] = {}])
(30)
```

The last instruction shows no functional dependence in the equations for the auxiliary fields (they were eliminated), and only $\{\phi\}$ dependence for the ϕ equation. Finally, this $EQ1[\phi]$ equation is tested against EQ obtained from L :

```
> expand(EQ1[phi]-EQ);
0
```

arriving at the expected result, which states the equivalence between L and $L1$.

¹⁰The *left hand sides* identify the fields to which the movement equations are associated, while the *right hand sides* display the required information. The **type/Function** subroutine is also part of the *partials* package (see Appendix B).

¹¹The **Maple** mechanism used in this step is known as *simplify with respect to side relations* (by doing Grobner basis calculations).

3 Conclusions

It is clear that computer support for theoretical research in physics is going to grow in the following years. In this context, the introduction of handy and flexible commands for the evaluation of partial and functional derivatives is important since it extends the use of computer algebra systems in research.

The building of such commands was presented here, as the *partials* package, of which the most relevant ones are **pdiff** and **fdiff**. As a consequence, problems requiring the calculation of partial and functional derivatives can now be tackled using an almost trivial syntax (characteristic of general purpose programs), as was shown in the examples treated in Sec.1 and 2.

Nevertheless, we are still far from the powerful required tool. The main problem is that there still does not exist a handy manipulation of indices associated with different representations for the space-time and for common internal groups appearing in almost all interesting physical theories. Specifically, it is not yet possible to evaluate functional derivatives with respect to Grassman variables or to abstract indexed quantities in a direct manner, as one is used to doing by hand. These subjects are now under study and will be reported elsewhere.

Acknowledgements

The author wishes to thank Prof. S.A.Dias¹² and Prof. J.F.Medeiros¹³, for fruitful comments and careful reading of this paper.

Appendix A

This appendix contains a discussion about the use of *mappings* in the calculation of partial derivatives. To start with, let's recall that

- the objects with which the **Maple** program works are classified into *types*.

These *types*, in turn, resemble the natural classification we are used to giving to mathematical objects. To know how **Maple** classifies an object, one can use either the **whattype** or the **type** command. For example:

```
> whattype(t);
```

string

```
> type(f(t),function);
```

¹²From **C.B.P.F.**, Rio de Janeiro, Brazil.

¹³From **U.E.R.J.**, Rio de Janeiro, Brazil.

true

Objects which are identified as functions by the **type** command are not allowed as derivation variables for the **diff** command. Therefore, typical operations involving variational calculus are not possible using **diff** in a direct manner. The standard alternative to this restriction requires the use of mappings together with the **D** operator while the alternative proposed in this paper is a straight use of the **pdiff** command. To understand the standard alternative, let's first review some aspects of a **Maple** mapping.

Mappings may be used to represent explicit functions. For example, for

$$f(x, y, z) = x + y^2 - z, \tag{31}$$

one can introduce the mapping:

```
> f := (x,y,z) -> x + y^2 - z:
```

$$f := (x, y, z) \mapsto x + y^2 - z \tag{32}$$

The main properties of a mapping are:

- It can be evaluated at a “point”.

For example,

```
> f(x,x,x);
```

$$x^2 \tag{33}$$

- The (x, y, \dots) variables used in the building of a mapping are *local* to it.

Therefore, these *local* variables cannot have their *names* associated with *global* meanings, as is commonly done in physics. As regards to this, the *mapping* defined in eq.(32) does not represent the function of eq.(31) since (31) has an explicit (x, y, z) dependence, probably with geometrical meanings associated to (global) (x, y, z) , while (32) does not depend on (x, y, z) at all. As an example of this difference, it does not make sense to differentiate the mapping f defined in eq.(32) with respect to x , y or z but only with respect to its “first”, “second” or “third” argument.

Now, as for the restriction mentioned at the beginning of this appendix, to “differentiate with respect to functions”, the User must build a *mapping*, use the **D** operator and indicate the *numeric position* of the derivation variable, as well as the *point* at which the result must be evaluated. This *evaluation point*, in turn, may contain functions. In this manner, one will receive what would be a representation for “differentiation with respect to functions”. As an example, the derivative of f with respect to the “second variable”, evaluated at the (function) point $(y, z(t), x)$, returns:

```
D[2](f)(y,z(t),x);
```

$$2z(t) \tag{34}$$

and, if x , y and $z(t)$ appear inside the evaluation point in the correct order (not the case above), the result may be interpreted as differentiation of eq.(31) w.r.t the function $z(t)$. Finally,

- objects which are identified as functions by the **type** command are not allowed as the (x, y, \dots) variables used in the building of a mapping.

Hence, *composite functions* may not have a simple “mapping representation”.

From the items above, the writer’s opinion is that mappings are very powerful objects, but their use for the calculation of partial derivatives implies on undesired complications and restrictions. In the case of partial derivatives of composite mappings, it is easy to make mistakes related to the wrong numeric position of each variable inside each mapping, or related to the order of the variables in each evaluation point. Besides, the building of the composite mappings may be not obvious and, in many situations, it may be convenient to have global meanings associated to the variables.

The **pdiff** command of the *partials* package presented in this paper was proposed as an alternative to this restrictions by accomplishing the task of partial derivatives of any type, in a simpler manner, as was shown in sections 1 and 2.

Appendix B

Making a slight change to the standard **D/procedure** routine of **Maple 5.2** and building some subroutines were necessary for the correct working of the commands of the *partials* package. The change is related to the following result and message:

```
> f(x,y,z) := x*y*z:      # Assigning a value to a function (not a mapping)
> D[2](f)(x,y,z);      # "D" returns an undesired result:
```

```
Error, (in D/procedure) index out of range: function takes only 0 arguments
(35)
```

This message is related to the automatic creation of a procedure of “0” arguments, which happens when one assigns values to a function. When a command of the *partials* package is loaded, the *Error* behavior mentioned above is automatically changed by inserting a few lines in the standard **D/procedure** routine. With the *partials* routines one receives:

```
> D[2](f)(x,y,z);      # Returns unevaluated as in Maple 5.1
                        D[2](f)(x,y,z)
(36)
> D[4](f)(x,y,z);      # Checks the number of arguments of the
                        # automatically created procedures by looking
                        # at their remember tables
```

```
Error, (in D/procedure) Function f takes no more than 3 arguments
```

Aside from the difference mentioned above, the **D/procedure** routine of the *partials* package works exactly as the standard one, and the unique case which turn apparent the difference is that which happens when one *assigns* values to a function which was not previously defined as a mapping.

Some **type** subroutines, built for the commands of the *partials* package, also useful by themselves are:

- **type/diff**, **type/Diff**, **type/De**, **type/Int**, **type/int**, and **type/Function**.

They allow a quick identification of derivatives, integrals, and functions (which are not derivatives or integrals or sums or limits), respectively.

Finally, the **Has** command analyze the functional dependence of algebraic expressions from a mathematical point of view, that is, discarding the dependence through integration variables of definite integrals. It was originally prepared as a subroutine for **pdiff** and can be easily extended in order to discard also summation indices, limit variables etc...

References

- [1] Michael Wester and Stanley Steinberg, A Survey of Symbolic Differentiation Implementations, *Proceedings of the 1984 MACSYMA Users' Conference*, Schenectady NY, (1984).
- [2] Gaston Gonnet, An Implementation of Operators for Symbolic Algebra Systems, *Proceedings of the 1986 Symposium on Symbolic and Algebraic Computations*, Symsac '86, ACM, (1986).
- [3] J.Barcelos-Neto, Ashok Das and W.Scherer, *Canonical quantization of constrained systems*, Acta Physica Polonica, Vol. **B18** No. 4 (1987) 269.
- [4] J.Barcelos-Neto and E.Vasquez, *Symplectic quantization with higher derivatives*, to appear in Zeitschrift für Physik C (1994).