diffcoeffx
extending the **diffcoeff** package

Andrew Parsloe
(aparsloe@clear.net.nz)

June 27, 2016

Abstract

`diffcoeffx.sty` is `diffcoeff.sty` 'on steroids'. It provides additional functionality for the trailing optional argument and extends the algorithm used to calculate the overall order of differentiation of mixed partial derivatives. That now accepts order-of-differentiation specifications that include powers of numbers and variables, subscripts on variables, and (possibly nested) parentheses with numerical coefficients. The enhancements come under the category of 'gilding the lily'.

1 The `diffcoeffx` package

The `diffcoeffx` package is `diffcoeff` 'on steroids', providing exactly the same commands but with some extra functionality. It is called in the usual way in the LaTeX preamble:

```
\usepackage{diffcoeffx}
```

It is assumed that you are familiar with the `diffcoeff` package and its manual. There are two enhancements to that package: `diffcoeffx` takes the calculation of the overall order of mixed partial derivatives deep into 'overkill' territory, accepting single-token powers of numbers and variables, single-token subscripts on variables, and possibly nested parentheses with numerical coefficients. The `\times` token ($\times$) can also be used in an order specification. The other enhancement is an extension to the capabilities of the trailing optional argument.

1.1 Exploiting the trailing optional argument

For `diffcoeff` there was an attempt to give a 'natural feel' to the design choices made and their use. By comparison the additional functionality that the trailing optional argument acquires in `diffcoeffx.sty` is in the nature of a *hack*. It works, but I'm not sure that it should be encouraged.

In `diffcoeff` if you write `\diff yx{}` the trailing but *empty* optional argument is ignored. Not so in `diffcoeffx`:

$$\texttt{\textbackslash diffp yx\{\}} \Longrightarrow \quad \left(\frac{\partial y}{\partial x}\right)$$

The parentheses are inserted without a subscript. Thus we can write (for instance) Lagrange's equations of motion in analytical mechanics in the manner:

`\diffp L{q_k}-\diff*{\diffp L{\dot{q}_k}{}}t = 0` $\Longrightarrow \quad \frac{\partial L}{\partial q_k} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_k}\right) = 0,$

without having to bother with inserting `\left(` and `\right)`. The empty trailing optional argument and the default delimiters for partial derivatives do the job for us.

There are many other places in analytical mechanics where using an empty trailing optional argument is a similarly convenient way of writing large parentheses, for instance,

`\dot{q_k}=\diffp H{\diffp S{q_k}{}}` $\Longrightarrow \quad \dot{q_k} = \frac{\partial H}{\partial\left(\frac{\partial S}{\partial q_k}\right)}.$

An application of Lagrange's equations (to a one-dimensional elastic solid) gives rise to a Langrangian density function,

```
\frac 12\left\{ \rho\dot{\eta}^2-E\diff\eta x{;2;()}\right \}
```

$$\Longrightarrow \quad \frac{1}{2}\left\{\rho\dot{\eta}^2 - E\left(\frac{d\eta}{dx}\right)^2\right\}.$$

Another application of those equations (the acoustic approximation to the irrotational motion of a compressible non-viscous fluid) produces a Lagrangian density

```
\frac 12\rho\left\{(\nabla\psi)^2-\frac 1{c^2}\diff\psi t{;2;()}\right\}
```

$$\Longrightarrow \quad \frac{1}{2}\rho\left\{(\nabla\psi)^2 - \frac{1}{c^2}\left(\frac{d\psi}{dt}\right)^2\right\}.$$

In both examples, the trailing optional argument of the *ordinary* derivative has been filled by a semicolon-delimited list: `{;2;()}`. The initial slot where a subscript is specified is empty but the semicolon is necessarily included. The second spot specifies a *superscript* and the third slot the delimiters to use. Since parentheses are not the default delimiters for an ordinary derivative we needed to specify them explicitly here. However, this does not change the default delimiters which remain `.|` for an ordinary derivative and can only be changed by means of the `\diffset` command.

Both subscript and superscript can be used at the same time. In a text on ordinary differential equations, an example employing Green's functions gives rise to

`\diff[n-1]Gx{\xi-\epsilon;\xi+\epsilon;[]}` $\Longrightarrow \quad \left[\frac{d^{n-1}G}{dx^{n-1}}\right]_{\xi-\epsilon}^{\xi+\epsilon}$

the derivative being evaluated at both superscript and subscript values and the difference taken. Here the trailing optional argument has its first three slots filled, with square brackets explicitly specified. The same book includes the example

`\diff*[p-1]{x^\alpha}\alpha{\alpha=a;;\{\}}` $\Longrightarrow \quad \left\{\frac{d^{p-1}}{d\alpha^{p-1}}x^\alpha\right\}_{\alpha=a}$

where, this time braces are specified in the trailing optional argument.[1]

This argument can be used to form the absolute value of a derivative,

$$\texttt{\textbackslash diff yx\{;;||\}} \Longrightarrow \quad \left|\frac{dy}{dx}\right|$$

where both initial slots, subscript and superscript, are empty and two semicolons necessarily included in the trailing optional argument: `{;;||}`. It also provides an alternative way, indeed *two* alternative ways, of forming a quotient of derivatives:

$$\texttt{\textbackslash diff yx\{;;./\}\textbackslash diff xy=\textbackslash diff yx\textbackslash diff xy\{;;/.\}} \Longrightarrow \quad \left.\frac{dy}{dx}\right/\frac{dx}{dy} = \frac{dy}{dx}\left/\frac{dx}{dy}\right.$$

where the delimiter specification `./` on the left has been changed to `/.` on the right. The spacing in the two quotients is not quite identical, which might be relevant in some contexts. As a more realistic example of use of the same construct, if $F(x,t)$ is a function of x and t and $x = x(t)$, then if $\frac{dF}{dt} = 0$,

$$\texttt{\textbackslash diff xt=-\textbackslash diffp Ft\{;;./\}\textbackslash diffp Fx} \Longrightarrow \quad \frac{dx}{dt} = -\left.\frac{\partial F}{\partial t}\right/\frac{\partial F}{\partial x}$$

For an inline use, you may prefer to use the slash form of the derivative $(dy/dz)_0$. In this case a *fourth* slot in the trailing optional argument has been filled, the `nudge override` slot, since the default nudge is designed to position the subscript relative to the *displaystyle* delimiters.

The complete specification of what is available in the trailing optional argument is:

```
{ subscript; superscript; delimiters; nudge override }
```

- In 'normal' use, the `subscript` is the point of evaluation (ordinary derivatives), or list of variables held constant (partial derivatives). Since the list of variables held constant is likely to be comma-separated, so we have the need for semicolons to separate items in the larger list.

- The `superscript` is generally a power to which the derivative is raised but, as instanced by the Green's function example, it can also be another point of evaluation of the derivative.

- The `delimiters` are, by default, `.|` for ordinary derivatives and `()` for partial derivatives. These are not always the right ones for a particular task. Rather than changing them *globally* as the use of `\diffset` entails, they can be changed *locally* for the particular instance by specifying them in this slot. The global choices are unaffected.

- If the built-in placement of sub- or superscript relative to the right delimiter is unsatisfactory, a value specified in the `nudge override` slot overrides the default value locally. The value is a pure number which `diffcoeffx` treats as that number of mu (1/18 of an em). (For comparison, a thin space \, and a negative thin space \! are 3/18 of an em.) The default nudges are shown in Table 1. They are intended for displaystyle presentation, and are not affected by any value included in this slot.

[1] For LyX users, the braces \{ and \} are inserted into a formula in the maths editor simply by typing the braces without the backslashes. LyX takes care of the latter.

Note that if one wants to use the nudge override with the default delimiters, it is necessary to indicate all preceding slots, even if they are empty, e.g., `{;;;-3}`. Similarly, to change the delimiters, to parentheses say, without sub- or superscript, it is necessary to indicate all preceding empty slots, but the following one does not need to be indicated: `{;;()}`. If one wants to specify a superscript, 2 say, but leave all else unchanged, it is only necessary to specify the one preceding empty slot: `{;2}`. Trailing empty slots can be omitted, which is why, if one wants to use the trailing empty argument 'as nature intended', i.e., to specify a point of evaluation or variables held constant, one can close one's mind to the other potential slots and simply write (for instance) `{0}` or `{x=1}`.

Table 1: Default nudges

right delimiter	nudge
), >	-6
\}	-4
|,]	0
other	0

1.2 The enhanced mixed partial derivative algorithm

In the documentation for `diffcoeff.sty` I discussed the transition table, Table 3a, in which signed **s**, numeric **n**, or algebraic **a** states changed to one of the others, or not, depending on the nature of the current token: sign, digit or variable. Signs and digits were explicitly defined; anything and everything else was called a (prime) variable. (Not quite true: in fact `diffcoeff.sty` checked for `(`, `^` and `_` and raised an error if they were encountered.)

Table 2: A first enhancement

	Curr. state	Curr. token	Action	Next state
1	**s**	s	$Ts \to s'$; $T = s'$	**s**
2	**s**	d	Td	**n**
3	**s**	v	Vv; $T1v$	**a**
4	**n**	s	$\mathbf{N}T$; $T = s$	**s**
5	**n**	d	Td	**n**
6	**n**	v	Vv; Tv	**a**
7	**a**	s	$\mathbf{V}V,$; $V = \emptyset$; $\mathbf{A}T$; $T = s$	**s**
8	**a**	d	error	**!!**
9	**a**	v	Vv; Tv	**a**

(a) State transitions

	Curr. state	Curr. token	Action	Next state
8	**a**	d	Vd; Td	**a**

(b) Allowing powers of variables

There is a certain inner logic at play here. Multi-token variables like kmn are included in the above scheme. But having accommodated mn, surely one should be able to handle mm, i.e. m^2? And if m^2, then why not m^n? In fact it is easy to do so. Since the superscript token ^ is neither sign nor digit, no longer raise an error if it is encountered but treat it, among the 'everything else' tokens, as a variable. If we change row 8 of the table as in Table 3b we have enlarged our scheme to include powers of variables – not only numerical powers (row 8) but also algebraic powers (row 9). As a side-effect, if we also suppress the raising of an error when the subscript token `_` is encountered, it too will be classified as a variable and allow numeric and algebraic subscripts on variables: things like k_2 or k_n.

Implicit in this discussion is the understanding that exponents and subscripts are restricted to *single tokens*. Coping with multi-token quantities in those positions would entail changes to other parts of the code, which I have chosen not to do.

This is a simple way of enlarging the range of tokens acceptable to the overall-order algorithm, but it does assume that the user does *not* include a sign as a superscript or subscript. If they do, then when the algorithm meets the sign it arrives at row 7 of the table and stores what is clearly an unintended variable, something like `k^` or `k_`. So, we need to check when a sign is met whether the previous token was one of `^` or `_` and raise an error if it was. But then the thought arises: if we are going to the trouble of checking for sub- or superscript tokens, why just raise an error? Why not incorporate signs in sub- or superscript positions into the scheme?

To this end, we might introduce a fourth state, the *script* state, denoted by **p**. A script token, denoted p, is one of `^` or `_`. There is only one way to enter a script state, and that is by appending a script token to a *variable*. Appending a script token to a sign or number (or, indeed, another script token) raises an error. Table 3 is the result. In this scheme, signs can be used as sub- or superscripts to variables, but not to numbers. We might console ourselves with the thought that this is, in any case, a limitation of the calculational engine used to evaluate our integer expressions. The `l3int` module of the LaTeX3 bundle `l3kernel` cannot handle powers of integers.

Yet this still leaves an unfinished feeling. While attaching a script token to a sign or other script token is a nonmathematical usage, attaching a superscript token to a number is a basic mathematical use, and so two of the errors raised can really be ignored. For the other, the question nags: why should we have to remember that although variables can be raised to powers, numbers cannot be? The urge to enlarge the scheme again is irresistible. Exponents on numbers should be accepted; but subscripts should not. The latter is a nonmathematical usage or is used only in special contexts remote from the present one. But that means we need to distinguish sub- and superscript tokens. We can't lump them together as 'script tokens'.

Raising numbers to powers: new states

So a first step is to enlarge the number of states. We need an *exponent* state **e** when we encounter the token ^ and a *subscript* state **b** when we encounter the token _. That allows us to distinguish acceptable forms like `2^3` (2^3) from unacceptable ones like `2_3` (2_3). But how do we know which state to transition to when we meet the `3` in `2^3`? The current state is the exponent one **e** and the `3` could be decorating either a variable or a number. We need to know the *previous* state as well as the current one. If the previous state is numeric we

Table 3: Transition states for an enlarged scheme

	Curr. state	Curr. token	Action	Next state
1	**s**	s	$Ts \to s'$; $T = s'$	**s**
2	**s**	d	Td	**n**
3	**s**	v	Vv; $T1v$	**a**
4	**s**	p	error	**!!**
5	**n**	s	$\mathbf{N}T$; $T = s$	**s**
6	**n**	d	Td	**n**
7	**n**	v	Vv; Tv	**a**
8	**n**	p	error	**!!**
9	**a**	s	$\mathbf{V}V,$; $V = \emptyset$; $\mathbf{A}T$; $T = s$	**s**
10	**a**	$x \in \{dv\}$	Vx; Tx	**a**
11	**a**	p	Vp; Tp	**p**
12	**p**	$x \in \{sdv\}$	Vx; Tx	**a**
13	**p**	p	error	**!!**

transition to a numeric state; if it is algebraic, we transition to an algebraic state.

But that also introduces a problem. It is perfectly acceptable to add a digit to a term in a numeric state. Normally, this is how a multi-digit number is accumulated: `234`, two hundred and thirty four. That is a very different meaning from `2^34` which means 2^34 to us (rather than 2^{34} since we accept only single-token superscripts). At this point, the syntax required by the underlying engine used for evaluating numerical expressions comes into play. For all numerical evaluations except those involving exponents, `l3int` of the LaTeX3 kernel is used; for expressions involving exponents, `l3fp` is used. To `l3fp`, `2^34` is read as 2^{34}. We need to insert a multiplication token between the `3` and `4`, which for `l3fp` is the asterisk, `*`. Considering the different tokens that might follow *that*, we are forced to introduce a third new state, the *multiplicative* state, **m**. So, to introduce powers of numbers means considering three new states and reference to the previous state.

That, of course, is *numeric* powers of numbers. To also allow algebraic powers, forms like 2^n, introduces further complication. These can't be evaluated numerically, so presumably they are to be classified as variables. We need to consider terms like $+2^n$, $3*2^n$, 3^m2^n, and 2^nm. The problem here is that we have something that looks as if it is going to be a number (the digit 2) but then transforms into a variable, 2^n. Do we need a *fourth* new state, the entangled state **q** (the 'q' as in 'quantum entanglement')?

In fact I find that these problems can all be dealt with not by creating another state but by including another accumulator for *potential* variables. I'll call it Q (from the quantum suggestion, or perhaps quasi-variable, or even acqumulator). It stores numbers whose status has not been determined yet: they might yet be followed by a superscript token which might in turn be followed by a variable. Once resolved, Q either transfers its contents to V, the

Table 4: State transitions of the full scheme

	S_-	S	$t \in \{sdv\hat{}\_*\}$	Action	S_+
1		**s**	s	$Ts \to s'$; $T = s'$	**s**
2		**s**	d	Qd; Td	**n**
3		**s**	v	Vv; $T1v$	**a**
4		**n**	s	$Q = \emptyset$; $\mathbf{N}T$; $T = s$	**s**
5		**n**	d	Qd; Td	**n**
6		**n**	v	$Q = \emptyset$; Vv; Tv	**a**
7		**n**	$\hat{}$	$Q\hat{}$; $T\hat{}$	**e**
8		**n**	$*$	$Q = \emptyset$; $T*$	**m**
9		**a**	s	$\mathbf{V}V,$; $V = \emptyset$; $\mathbf{A}T$; $T = s$	**s**
10	**e**	**a**	d	Vd; Td	**a**
11		**a**	v	Vv; Tv	**a**
12		**a**	$\hat{}$	$V\hat{}$; $T\hat{}$	**e**
13		**a**	$\_$	$V\_$; $T\_$	**b**
14	**a**	**e**	$t \in \{sdv\}$	Vt; Tt	**a**
15	**n**	**e**	d	$Q = \emptyset$; $Td*$	**m**
16	**n**	**e**	v	Qv; Tv; $V = Q$; $Q = \emptyset$	**a**
17	**a**	**b**	$t \in \{sdv\}$	Vt; Tt	**a**
18	**e**	**m**	s	$T1$; $\mathbf{N}T$; $T = s$	**s**
19		**m**	d	Qd; Td	**n**
20		**m**	v	Vv; $T1v$	**a**
21		**m**	$*$		**m**

variable accumulator, and is emptied (row 16), or is emptied forthwith (rows 4, 6, 8, 15).

Table 4 lists the transitions. I've denoted the previous state by S_-, the present state by S, and the next state by S_+. The final row of the table is intended: do nothing if we meet a multiplicative token when in a multiplicative state. The first scan through an order specification (to split it into numeric and algebraic parts) may introduce a * token (rows 8 and 15). We don't want to introduce a second such token in the recursive determination of the coefficients of variables. Hence row 21: do nothing. Also, if in the order specification we have something like $2^3 * 3^2$ (since `2^33^2` looks weird), we don't want the manually inserted * to cause an error because of the automatically inserted one (row 15).

Possibilities not explicitly present in the table generally raise an error, e.g. current state **s** and current token ^, or previous state **n**, current state **e** and current token s ($+$ or $-$), and so on. I have omitted them from the table in the interests of space. The table is big enough already.

With this table of transitions it is now possible to handle order specifications that include components like n^2 or n^m or k^+ or k_2 or k_n or 2^2 or 2^33^2 or 2×3^n or 2^23^n or ...

Note that the `\times` token is converted internally by `diffcoeffx.sty` to the asterisk. They can be used interchangeably but it certainly looks more elegant.

So, what could be better on a cool winter's evening, snug before the warmth of the fire, a glass of sustaining liquid to hand, than to do a few mixed partial derivatives? Like this,

```
\diffp[3^22^22^n+m,12^n-3m+2^3k,5m+2\times2^32^n]{F(x,y,z)}{x,y,z}
```

$$\Longrightarrow \frac{\partial^{52\times 2^n+3m+8k}F(x,y,z)}{\partial x^{3^22^22^n+m}\,\partial y^{12^n-3m+2^3k}\,\partial z^{5m+2\times 2^32^n}}$$

or like this,

```
\diffp[k^+k_-+1,2\times k_-,3^2k_-,3k^+]{F(x,y,z,w)}{x,y,z,w}
```

$$\Longrightarrow \quad \frac{\partial^{k^+k_-+11k_-+3k^++1}F(x,y,z,w)}{\partial x^{k^+k_-+1}\,\partial y^{2\times k_-}\,\partial z^{3^2k_-}\,\partial w^{3k^+}}$$

In the first example the `\times` symbol is inserted by `diffcoeffx` in the overall order of differentiation in the numerator so as to prevent the formation 522^n which would be read as 522 raised to the power n – and for a similar reason it was used in specifying the order of differentiation of the variable z in the denominator in the first example, but could and should have been deleted from the order of differentiation of the variable y in the second example.

Parentheses

The other major shortcoming of the basic scheme outlined in `diffcoeff.sty` was the inability to handle even the simplest instance of parentheses in an order specification – something like `[m-(n-1),m+(n-1)]` which might well arise in a Taylor expansion. Indeed, there is more reason for including these in our scheme than exponents of numbers or $+$ or $-$ as sub- or superscripts.

How might we fit parentheses to the scheme? We are not seeking a general treatment. Rather we wish to be able to handle order specifications a little more complicated (but only a little) than the one just given, say something like `[m+2(n-1),m-(n-1)]`, perhaps with nesting. In that case the following stipulations meet our needs:

Table 5: Parentheses

	Curr. state	Curr. token	Action	Next state
1	**s**	(	$T1{*}($; $\mathbf{N}T$; $\mathbf{A}T$; $T=+$	**s**
2	**s**	)	$\mathbf{N})$; $\mathbf{A})$	**s**
3	**n**	(	$T*($; $\mathbf{N}T$; $\mathbf{A}T$; $T=+$	**s**
4	**n**	)	$T)$; $\mathbf{N}T$; $\mathbf{A}{+}0)$; $T=+$	**s**
5	**a**	)	$\mathbf{V}V,$; $V=\emptyset$; $\mathbf{N}{+}0)$; $T)$; $\mathbf{A}T$; $T=+$	**s**
6	**m**	(	$T($; $\mathbf{N}T$; $\mathbf{A}T$; $T=+$	**s**

- a left parenthesis, (, either starts an item in the comma list, or is preceded by a sign or a number or $*$ or (, but *not* by a variable or ^ or _ or);
- a right parenthesis,), either concludes an item in the comma list, or is followed by a sign or), but *not* by a number or a variable or ^ or _ or $*$ or (.

These limitations allow nesting of parentheses but not products of parentheses. The main limitation they impose is that a variable lie *within* parentheses but not adjoining-outside. They enable us to get away with the following 'cheap and cheerful' scheme. It means we do not need to add parenthesis states to our scheme. The particular point to note are the +0 insertions. When we start parsing an expression from the left we do not know what it contains. In particular when we meet a left parenthesis, we have no foreknowledge of whether the parenthesised expression will be numeric, algebraic or a mix of both. We need to prepare for both by inserting a left parenthesis to both numeric and algebraic parts. But that brings us up against a quirk of `l3int`, the 'engine' behind the numerical evaluations performed in `diffcoeff` and `diffcoeffx`. `l3int` objects to an empty pair of parentheses, `()`, which we would have should either numeric or algebraic parts be missing from the parenthesised expression. To avoid this we insert +0 and `l3int` is happy.

- Row 1. Quirks of the `l3int` module of the LaTeX3 kernel mean we need to insert **1*** before the left parenthesis.[2] Note that we add T to *both* the numeric and algebraic parts of the expression. We are working through our expression **E** from the left, token by token, and have no foreknowledge of what the parenthesised expression contains, whether algebraic terms only or numeric terms only or some combination of both. Hence the need to prepare for both. The system shifts to a signed state **s** with $T=+$, exactly the same as when beginning to scan **E**. After all, the parenthesised expression is an expression in itself.
- Row 2. This is to allow nested parentheses like)). It shouldn't arise otherwise. Because of rows 4 and 5, the first right parenthesis puts the system into a signed state. The current term will be $T=+$, but we ignore it and store only a right parenthesis in both numeric and algebraic parts.

[2]Specifically, `\int_eval{ -(` or `\int_eval{ +(` throw errors.

- Row 3. We already have a number present in T; only the asterisk needs inserting before the parenthesis. Again we add T to *both* the numeric and algebraic parts of the expression, initialise T to + and change the state to a signed one.
- Row 4. We are in a numeric state. We append) to the current term and the current term to the numeric part of the expression. We append +0) to the algebraic part, and shift to a signed state **s** with $T = +$, as at the outset. The +0) in the algebraic part is necessary to prevent an empty parenthesis pair in **A** should the parenthesised expression have contained *no* algebraic term.
- Row 5. We are in an algebraic state. We append) to the current term and the current term to the algebraic part of the expression. We append +0) to the numeric part and shift to the initial signed state again. The +0) in the numeric part is necessary to prevent an empty parenthesis pair in **N** should the parenthesised expression have contained *no* numeric term.
- Row 6. We are in the new state, the multiplicative state, and the current token is a left parenthesis. We have already met and inserted an asterisk (row 5); we don't need to insert another. We append (to T, T to both numeric and algebraic parts and shift to the initial signed state again. The use of the multiplicative state prevents a string of asterisks arising (but we have been able to avoid introducing new states for left and right parentheses).

To work through an example, suppose we have an order specificiation **[m+(n–(k–1)),m–2(n+(k–1)),k]**. (Digit **1** rather than a lower-case letter **l** within the parentheses!) Concatenating, with linking + signs gives **m+(n–(k-1))+m–2(n+(k–1))+k**. Splitting into numeric and algebraic parts now results in **+1*(–1*(–1))–2*(+1*(–1))** for the numeric part, evaluating to **3**, and **+1m+1*(+1n–1*(+1k+0))+1m–2*(+1n+1*(+1k+0))+1k** for the algebraic part.

Removing **m** from the latter and splitting into numeric and algebraic parts gives **+1+1*(–1*(+0))+1–2*(+1*(+0))** for the numeric part, evaluating to **2** which is the overall coefficient of **m**, and **+1*(+1n–1*(+1k+0))–2*(+1n+1*(+1k+0))+1k** for the algebraic part.

Now remove **n** from this resulting algebraic part and again split into parts. The result is **+1*(+1-1*(+0))–2*(+1+1*(+0))** for the numeric part, evaluating to **–1** which is the overall coefficient of **n**, and **+1*(–1*(+1k+0))–2*(+1*(+1k+0))+1k** for the algebraic part.

Removing **k** from this and splitting gives **+1*(–1*(+1+0))–2*(+1*(+1+0))+1** for the numeric part, evaluating to **–2** which is the overall coefficient of **k**, and **+1*(–1*(+0)) –2*(+1*(+0))** for the algebraic part. But we have run out of variables and so the process stops at this point:

$$\texttt{\textbackslash diffp[m+(n-(k-1)),m-2(n+(k-1)),k]F\{x,y,z\}}\Longrightarrow \frac{\partial^{2m-n-2k+3}F}{\partial x^{m+(n-(k-1))}\,\partial y^{m-2(n+(k-1))}\,\partial z^{k}}$$