Manual Math. Inst., Univ. Leipzig (1997) , 1–22

VECTAN 1.1

Bernd Fiedler

Department of Mathematics, University of Leipzig

Augustusplatz 10/11, D-04109 Leipzig, Germany

email: fiedler@rz.uni-leipzig.de

http://www.mathematik.uni-leipzig.de/MI/fiedler/fiedler.html

(May 1997)

VECTAN 1.1 is a Mathematica package for doing calculations of vector analysis. It was developed for the treatment of certain special problems and works good in these cases. Other situations will possibly show the need of the extension of VECTAN. The development of this program is not yet finished.

VECTAN is a good example for a programming based on rules. However, it does not contain special normal form algorithms for expressions which occur in vector analysis. From this point of view, VECTAN is only a first step on the way to a more extensive package concerning vector analysis.

Please, send remarks, informations on bugs and suggestions for improvement to the author.

1. Copyright

2. Installation

All files in the archive vectan.tar.gz are MS-DOS text files with line end 0D 0A. If you use an other operation system you have possibly to convert these files by means of suitable tools (e.g. fromdos in Linux).

The files

$$\begin{aligned} &vectan.m \\ &def.m \\ &deriv.m \\ &usage.m \\ &utilitie.m \end{aligned} \tag{2.1}$$

contain the English version of VECTAN. If you wish to use the German usage texts, rename the files vectan.ger and usage.ger to vectan.m and usage.m and replace the corresponding files of the English version by these.

The print format of VECTAN for the vector product $a \times b$ uses a special character for the sign '$\times$', which is stored in the variable $VPChar. The default value of $VPChar is the value for MS-DOS. If you wish to run VECTAN under another operating system you can give $VPChar a suitable default value by editing the definition at the end of the file usage.m.

Now you can install VECTAN by the following two steps:

1. Copy all files (2.1) in a separate directory 'dir'.
2. Change to the directory 'dir' (working directory), run Mathematica and load VECTAN by '$<< vectan.m$'.

3. Functions and vector fields

Functions and vector fields are considered to be quantities which depend on variables t, x_1, x_2, x_3. Operators like grad, div, rot, dot,... acts on these quantities. Grad, div, rot are determined from derivatives according to x_1, x_2, x_3 whereas dot denotes the derivative according to t. The dependence of functions and vector fields on t, x_1, x_2, x_3 becomes nowhere visible in a calculation by means of VECTAN. VECTAN works on expressions like

$$div[(grad[f] \times b) \times c] \quad ,$$

but it does not work on expressions like

$$\frac{\partial f}{\partial x_1} + \frac{\partial f}{\partial x_2} + \frac{\partial f}{\partial x_3} \quad .$$

In VECTAN a function f can be defined simply by generating a Mathematica symbol f :

```
In[10] := f

Out[10] := f

In[11] := ?f
Global`f
```

Table 1. Operators and products in VECTAN.

input form	print form	arguments	explanation
$SP[a,b]$	$< a,b >$	vectors	scalar product
$VP[a,b]$	$a \times b$	vectors	vector product
$grad[f]$	$grad[f]$	function	gradient
$grad[a,b]$	$(< a, grad >)[b]$	vectors	derivative of b in the direction of a
$rot[a]$	$rot[a]$	vector	curl
$div[a]$	$div[a]$	vector	divergence
$dot[u]$	a'	function, vector	derivative according to t

```
In[11] :=
```

A vector a field in VECTAN is a Mathematica object Vect[a]. Vect is a head defined in VECTAN, which has the attribute HoldFirst. Vector fields can be defined in VECTAN by means of DefineVector:

```
In[3] := DefineVector[a]

DefineVector::inf: Vector field a has been established.

In[4] := ?a
Global`a

a = Vect[a]

In[4] :=
```

Several vector fields can be defined by means of DefineVector at the same time if one applies DefineVector to a list of symbols for vector fields.

```
In[5] := DefineVector[{b, c, d}]

DefineVector::inf: Vector field b has been established.

DefineVector::inf: Vector field c has been established.

DefineVector::inf: Vector field d has been established.

In[6] :=
```

4. Operators

Table 1† gives a list of operators and products, which are available in VECTAN. The Laplace operator is not explicitely defined at the moment in VECTAN.

† The sign $\times$ in a vector product is printed as the character the code of which is stored in the variable $VPChar.

5. Definition of results for products and operator actions

VECTAN presents the possibility to define values for the results of certain products and for the results of the action of operators on functions and vector fields. Such definitions can be established by means of

$$\begin{array}{l} \text{divValue} \\ \text{dotValue} \\ \text{gradValue} \\ \text{graddotValue} \\ \text{rotValue} \\ \text{SPValue} \\ \text{VPValue} \quad . \end{array} \tag{5.1}$$

The following example shows, how the condition

$$rot[a] \quad = \quad c \times b$$

can be defined by the user:

```
In[3] := DefineVector[{a,b,c}]

DefineVector::inf: Vector field a has been established.

DefineVector::inf: Vector field b has been established.

DefineVector::inf: Vector field c has been established.

In[4] := a/: rotValue[a] := VP[c,b]

In[5] := ?a
Global`a

a = Vect[a]

rotValue[a] ^:= VP[c,b]

In[5] := rot[a]

Out[5] = -(b x c)
```

One reason for the use of the tools (5.1) is the fact that the FullForm of an expression like $rot[a]$ is $rot[Vect[a]]$. Thus an up-value-definition

```
a/: rot[a] := VP[c,b]
```

is not allowed in Mathematica. All tools (5.1) possess the attribute HoldAll such that

```
a/: rotValue[a] := VP[c,b]
```

is a correct up-value-definition in Mathematica. (See also the comments in the file def.m).

If a definition is created with a tool (5.1) which has more than one arguments, the definition can be assigned to one of the arguments. E.g. we have the possibilities

```
a/: SPValue[a,b] := 0
```

or

```
b/: SPValue[a,b] := 0          .
```

All definitions which are connected with symbols in the actuell context can be saved by means of VASave and reloaded by means of Get in a later session.

Some of the tools (5.1) permit a set of composite patterns for the creation of definitions. An information on the allowed patterns can be obtained by means of the command Mu.

If a definition is established which involves further definitions, these definitions have been explicitly produce by the user too, that VECTAN shows a useful behaviour. For instance if we use the condition

```
f/: gradValue[f] := a
```

but forget to create the condition

```
a/: rotValue[f] := 0          ,
```

we do not obtain $rot[grad[f]] = 0$ in a calculation with VECTAN.

```
In[3]:= DefineVector[a]

DefineVector::inf: Vector field a has been established.

In[4]:= f

Out[4]= f

In[5]:= rot[grad[f]]

Out[5]= 0

In[6]:= f/: gradValue[f] := a

In[7]:= rot[grad[f]]

Out[7]= rot[a]

In[8]:= a/: rotValue[a] := 0

In[9]:= rot[grad[f]]
```

```
Out[9]= 0
```

Those tools (5.1) which possess more than one argument can be used to define patterns with variable arguments. (See usage text on graddotValue.)

```
In[3]:= DefineVector[{a,b,c,d}]

DefineVector::inf: Vector field a has been established.

DefineVector::inf: Vector field b has been established.

DefineVector::inf: Vector field c has been established.

DefineVector::inf: Vector field d has been established.

In[4]:= a/: gradValue[x_,a] := 0

In[5]:= grad[b,a]

Out[5]= 0

In[6]:= grad[c,a]

Out[6]= 0
```

6. The problem of normal forms

Calculations of vector analysis may lead to situations in which there are several possibilities to transform the present expression. For example, expressions like

$$(a \times b) \times (c \times d)$$
$$(a \times (b \times (c \times (d \times e))))$$

can be converted to other expressions by means of the so-called bac-cab-rule

$$a \times (b \times c) \quad = \quad < a, c > b - < a, b > c \tag{6.1}$$

in very different ways. Further the immediate situation decides whether a transformation like

$$\begin{aligned} < a, b \times c >< u, v \times w > \quad &= \quad & (6.2) \\ - < a, w >< b, v >< c, u > \quad &+ \quad < a, v >< b, w >< c, u > + < a, w >< b, u >< c, v > & \\ - < a, u >< b, w >< c, v > \quad &- \quad < a, v >< b, u >< c, w > + < a, u >< b, v >< c, w > & \end{aligned}$$

or

$$< a, b \times c > d \quad = \quad < c, d > (a \times b) - < b, d > (a \times c) + < a, d > (b \times c) \tag{6.3}$$

is usefull or not.

A solution of this problem would be given by algorithms which can change an arbitrary expression of vector analysis into a certain normal form. However, such algorithms are not included in VECTAN. Nevertheless, VECTAN has well carried out many calculations.

Table 2. The variables \$BacCab, \$KillSpat, \$SpatRule.

variable	default value	rule, automatically used in case True
\$BacCab	False	(6.1)
\$KillSpat	False	(6.3)
\$SpatRule	True	(6.2)

There are system variables \$BacCab, \$KillSpat and \$SpatRule in VECTAN, which control the behaviour of VECTAN relating to the rules (6.1), (6.2), (6.3) (see table 2). If such a variable has the value True, the corresponding rule given in table 2 is used automatically. The patterns which VECTAN has to identify in the case of automatical rule using are exactly of that form, which are applied in the left hand sides of (6.1), (6.2), (6.3). Such a rule is used in the automatic case whenever VECTAN detect a subexpression which matches the corresponding pattern of the rule.

If \$BacCab = False, the user can apply the bac-cab-rule by means of the tool BacCab (see the usage text on BacCab).

```
In[4]:= $BacCab

Out[4]= False

In[5]:= VP[VP[a,b],VP[c,d]]

Out[5]= ((a x b) x (c x d))

In[6]:= BacCab[%5, VP[a,b],c,d]

Out[6]= <a, (b x d)> c - <a, (b x c)> d

In[7]:= BacCab[%5, VP[b,a],c,d]

Out[7]= <a, (b x d)> c - <a, (b x c)> d

In[8]:= BacCab[%5, VP[c,d],a,b]

Out[8]= -(<b, (c x d)> a) + <a, (c x d)> b

In[9]:= VP[a,VP[b,VP[c,VP[d,VP[e,f]]]]]

Out[9]= (a x (b x (c x (d x (e x f)))))

In[10]:= BacCab[%9, b,c,VP[d,VP[e,f]]]

Out[10]= <b, (d x (e x f))> (a x c) - <b, c> (a x (d x (e x f)))
```

7. An example

In this section we give an example of the functioning of VECTAN. Extensive calculations of the type occuring in the example have been performed by means of VECTAN in investigations, the result of which are presented in the paper [1].

We consider two sequences $\{b_n\}_{n\geq 2}$, $\{d_n\}_{n\geq 2}$ of vector fields depending on the time t and on the spacial variables x_1, x_2, x_3. b_n and d_n have to be determined by means of the recursive rule

$$
\begin{aligned}
rot[d_n] &= \dot{b}_{n-1} + rot[r < n, a > d_{n-1}] && (7.1)\\
d_n &= rot[A_n] && (7.2)\\
d_{-2} &= \frac{e}{4\pi r^2} \quad , \quad b_{-2} = \frac{f}{4\pi r^2} \quad , \quad e, f = \text{const.} && (7.3)
\end{aligned}
$$

r denotes the function $r = \sqrt{(x_1)^2 + (x_2)^2 + (x_3)^2}$. We assume that the vector field a in (7.1) depends only on t. Consequently it fullfils the conditions

$$
\begin{aligned}
&div[a] = 0 \quad , \quad rot[a] = 0 && (7.4)\\
&\forall y : (< y, grad >)[a^{(m)}] = 0 \ , \ m = 0, 1, 2, ...
\end{aligned}
$$

The vector field n in (7.1) denotes the gradient of r, $n := grad[r]$. Thus n satisfies

$$
\begin{aligned}
&div[n] = \frac{2}{r} \quad , \quad rot[n] = 0 \quad , \quad \dot{n} = 0 \quad , \quad < n, n > = 1 && (7.5)\\
&\forall y : (< y, grad >)[n] = \frac{1}{r}\, y \; - \; \frac{< y, n >}{r}\, n \quad . && (7.6)
\end{aligned}
$$

All vector fields d_n are assumed to be depending on certain vector potentials A_n according to (7.2). Further we suppose that every vector field b_n can be obtained from the corresponding field d_n by means of the rule

$$
e \to f \quad , \quad f \to -e \quad . \qquad (7.7)
$$

The conditions (7.3), (7.4), (7.5), (7.6), (7.7) can be expressed by the following Mathematica statements, which are stored in the file startdat.m of the VECTAN archive.

```
(****************************** Definitions **************************)
$BacCab = True;
$SpatRule = False;

Attributes[e] = {Constant} ;
Attributes[f] = {Constant} ;
Attributes[z] = {Constant} ;


DefineVector[{a,n}];

a/: divValue[a] := 0 ;
a/: rotValue[a] := 0 ;
a/: gradValue[y_,a] := 0 ;
a/: graddotValue[y_,a,0,m_] := 0 ;
```

```
n/: divValue[n] := 2/r ;
n/: dotValue[n] := 0 ;
n/: rotValue[n] := 0 ;
n/: SPValue[n,n] := 1 ;
n/: gradValue[y_,n] := y/r - (SP[y,n]*n)/r ;

r/: dotValue[r] := 0 ;
r/: gradValue[r] := n ;

symrule = {e -> f, f -> - e} ;

b[-2] = (f n)/(4 Pi (r^2));
d[-2] = (e n)/(4 Pi (r^2));
```

We have the aim to construct special solutions of (7.1) for $n = -1$ and $n = 0$. For this purpose we make an ansatz for A_n

$$A_n \quad = \quad \sum_{k=1}^{m} z_{n,k} T_{n,k} \quad , \quad z_{n,k} = \text{const.} \tag{7.8}$$

The terms $T_{n,k}$ are certain expressions in a and n. For $n = -1, 0$ the concrete structure of these expressions is printed in the lines Out[8] and Out[30] bellow. Further these terms are stored in the files ansatz.n-1 and ansatz.n0. Now we construct the searched solutions of (7.1) by inserting the ansatzes (7.8) into (7.1), (7.2) and determining the unknown constants $z_{n,k}$.

At first we load VECTAN and the file startdat.m.

```
Mathematica 2.2 for DOS 387
Copyright 1988-93 Wolfram Research, Inc.

In[1]:= <<vectan.m

In[2]:= <<startdat.m
```

Then we consider the case $n = -1$. We load the terms $T_{-1,k}$ from the file ansatz.n-1, generate the corresponding unknown coefficients $z_{-1,k}$ and form the ansatz (7.8) for A_{-1}.

```
In[7]:= (* Step -1 *)

In[8]:= ansatzterms = <<ansatz.n-1

Out[8]= {a, <a, n> n, (a x n)}

In[9]:= vars = Array[z,Length[ansatzterms]]

Out[9]= {z[1], z[2], z[3]}

In[10]:= A[-1] = ansatzterms.vars
```

```
Out[10]= a z[1] + <a, n> n z[2] + (a x n) z[3]
```

After that we calculate the left hand side ls and the right hand side rs of (7.1) and determine ls - rs. The initial values of b_{-2} and d_{-2} had been delivered VECTAN by the loading of startdat.m.

```
In[11]:= ls = rot[A[-1]] //Expand

         (a x n) z[2]   a z[3]   <a, n> n z[3]
Out[11]= ------------ + ------ + -------------
              r           r            r

In[12]:= ls = rot[ls] //Expand

         2 <a, n> n z[2]   2 (a x n) z[3]
Out[12]= --------------- + --------------
                2                2
               r                r

In[13]:= b[-2]

           f n
Out[13]= -------
               2
         4 Pi r

In[14]:= d[-2]

           e n
Out[14]= -------
               2
         4 Pi r

In[15]:= rs = - dot[b[-2]] - rot[r SP[n,a] d[-2]] //Expand

         -(e (a x n))
Out[15]= ------------
                 2
           4 Pi r

In[16]:= gl = ls - rs

         e (a x n)   2 <a, n> n z[2]   2 (a x n) z[3]
Out[16]= --------- + --------------- + --------------
                2           2                  2
          4 Pi r           r                  r
```

Since the Mathematica tool Solve does not know the head Vect, which is used to define vectors u = Vect[u] in VECTAN, we can not determine the unknown constants z[1], z[2], z[3] by using the Mathematica statement Solve[ls - rs == 0, {z[1], z[2], z[3]}]. Mathematica would execute divisions by vectors.

We have to factor out vectors which occur in more than one summand of gl = ls - rs by means of Collect and to solve the equation which arises from the condition that the coefficient of such an vector has to vanish. We reinsert the value of every variable z[k] in gl, which we have determined. This reduces gl. The procedure is finished if gl vanishes identically. The result of this calculation is stored in the variable sol.

```
In[17]:= gl = Collect[gl, VP[a,n]]

         2 <a, n> n z[2]               e      2 z[3]
Out[17]= --------------- + (a x n) (------- + ------)
                2                         2       2
               r                      4 Pi r     r

In[18]:= part = Last[gl]

                            e      2 z[3]
Out[18]= (a x n) (------- + ------)
                        2       2
                    4 Pi r     r

In[19]:= sol = Solve[part == 0, z[3]][[1]]

                    -e
Out[19]= {z[3] -> ----}
                   8 Pi

In[20]:= gl = gl /. sol //Expand

         2 <a, n> n z[2]
Out[20]= ---------------
                2
               r

In[21]:= sol0 = Solve[gl == 0, z[2]][[1]]

Out[21]= {z[2] -> 0}

In[22]:= sol = Join[sol,sol0]

                    -e
Out[22]= {z[3] -> ----, z[2] -> 0}
                   8 Pi

In[23]:= gl = gl /. sol
```

```
Out[23]= 0
```

Now we insert the yielded values of the variables z[2], z[3] into the ansatz for A_{-1} and calculate d_{-1}. b_{-1} is obtained from d_{-1} by means of the rule (7.7). The variable z[1] is a free parameter.

```
In[24]:= A[-1] = A[-1] /. sol //Expand

         -(e (a x n))
Out[24]= ------------ + a z[1]
             8 Pi

In[25]:= d[-1] = rot[A[-1]] //Expand

         -(e a)   e <a, n> n
Out[25]= ------ - ----------
         8 Pi r     8 Pi r

In[26]:= symrule

Out[26]= {e -> f, f -> -e}

In[27]:= b[-1] = d[-1] /. symrule //Expand

         -(f a)   f <a, n> n
Out[27]= ------ - ----------
         8 Pi r     8 Pi r
```

The determination of A_0, d_0 and b_0 is performed in the same way. At first we build the ansatz for A_0.

```
In[28]:= (* Step 0 *)

In[30]:= ansatzterms = <<ansatz.n0

                                          2
Out[30]= {r a', r (n x a'), r <a, n>  n, r <n, a'> n, r <a, a> n,

>    r <a, n> a, -(r <a, n> (a x n))}

In[31]:= vars = Array[z,Length[ansatzterms]]

Out[31]= {z[1], z[2], z[3], z[4], z[5], z[6], z[7]}

In[32]:= A[0] = ansatzterms.vars

                                                      2
Out[32]= r a' z[1] + r (n x a') z[2] + r <a, n>  n z[3] +
```

```
>    r <n, a'> n z[4] + r <a, a> n z[5] + r <a, n> a z[6] -

>    r <a, n> (a x n) z[7]
```

Now we calculate gl = ls - rs according to (7.1).

```
In[33]:= ls = rot[A[0]] //Expand

Out[33]= (n x a') z[1] - 2 a' z[2] + 2 <a, n> (a x n) z[3] -

>    (n x a') z[4] -

                                                     2
>    2 <a, n> a z[7] + <a, a> n z[7] - <a, n>  n z[7]

In[34]:= ls = rot[ls] //Expand

           <n, a'> n z[1]     a' z[1]   2 <a, n> a z[3]
Out[34]= -(--------------) - ------- + --------------- -
                 r              r             r

     2 <a, a> n z[3]
>    --------------- +
            r

            2
     4 <a, n>  n z[3]   <n, a'> n z[4]   a' z[4]   4 <a, n> (a x n) z[7]
>    ---------------- + -------------- + ------- - ---------------------
            r                 r             r                r

In[35]:= rs = - dot[b[-1]] - rot[r SP[n,a] d[-1]] //Expand

         f <n, a'> n    f a'    3 e <a, n> (a x n)
Out[35]= ----------- + ------ + ------------------
           8 Pi r      8 Pi r        8 Pi r

In[36]:= gl = ls - rs

         -(f <n, a'> n)    f a'    3 e <a, n> (a x n)   <n, a'> n z[1]
Out[36]= -------------- - ------ - ------------------ - -------------- -
             8 Pi r       8 Pi r         8 Pi r                r

                                                             2
     a' z[1]   2 <a, n> a z[3]   2 <a, a> n z[3]   4 <a, n>  n z[3]
>    ------- + --------------- - --------------- + ---------------- +
        r             r                 r                 r
```

```
     <n, a'> n z[4]   a' z[4]   4 <a, n> (a x n) z[7]
>    -------------- + ------- - ---------------------
           r             r                r
```

We solve the equation gl = 0 for the variables z[1], z[2], ... , z[7] in the same way like in step -1.

```
In[38]:= gl = Collect[gl,n]

         -(f a')   3 e <a, n> (a x n)   a' z[1]   2 <a, n> a z[3]
Out[38]= ------- - ------------------ - ------- + --------------- +
         8 Pi r          8 Pi r            r             r

     a' z[4]
>    ------- +
       r

                                                              2
        -(f <n, a'>)   <n, a'> z[1]   2 <a, a> z[3]   4 <a, n>  z[3]
>    n (------------ - ------------ - ------------- + -------------- +
          8 Pi r            r               r                r

        <n, a'> z[4]    4 <a, n> (a x n) z[7]
>       ------------) - ---------------------
             r                    r

In[39]:= Length[gl]

Out[39]= 7

In[40]:= part = gl[[6]]


            -(f <n, a'>)   <n, a'> z[1]   2 <a, a> z[3]
Out[40]= n (------------ - ------------ - ------------- +
              8 Pi r            r               r

                2
       4 <a, n>  z[3]   <n, a'> z[4]
>      -------------- + ------------)
             r                r

In[45]:= sol0 = Solve[part == 0, z[1]][[1]]
```

```
Out[45]= {z[1] ->

                                                         2
      -(f <n, a'>) - 16 Pi <a, a> z[3] + 32 Pi <a, n>  z[3]
>     ------------------------------------------------------ +
                                  8 Pi <n, a'>

      8 Pi <n, a'> z[4]
>     ----------------- }
        8 Pi <n, a'>

In[46]:= gl = gl /. sol0 //Expand

         -3 e <a, n> (a x n)   2 <a, n> a z[3]   2 <a, a> a' z[3]
Out[46]= ------------------- + --------------- + ---------------- -
               8 Pi r                 r             r <n, a'>

              2
     4 <a, n>  a' z[3]   4 <a, n> (a x n) z[7]
>    ----------------- - ---------------------
         r <n, a'>                 r

In[47]:= sol = sol0

Out[47]= {z[1] ->

                                                         2
      -(f <n, a'>) - 16 Pi <a, a> z[3] + 32 Pi <a, n>  z[3]
>     ------------------------------------------------------ +
                                  8 Pi <n, a'>

      8 Pi <n, a'> z[4]
>     ----------------- }
        8 Pi <n, a'>

In[48]:= gl = Collect[gl,VP[a,n]]

                                                          2
         2 <a, n> a z[3]   2 <a, a> a' z[3]   4 <a, n>  a' z[3]
Out[48]= --------------- + ---------------- - ----------------- +
                r             r <n, a'>          r <n, a'>

              -3 e <a, n>   4 <a, n> z[7]
>    (a x n) (----------- - -------------)
                8 Pi r            r

In[49]:= part = Last[gl]
```

```
                  -3 e <a, n>   4 <a, n> z[7]
Out[49]= (a x n) (----------- - -------------)
                    8 Pi r            r

In[50]:= sol0 = Solve[part == 0, z[7]][[1]]

                  -3 e
Out[50]= {z[7] -> -----}
                  32 Pi

In[51]:= sol = sol /. sol0

Out[51]= {z[1] ->

                                                      2
      -(f <n, a'>) - 16 Pi <a, a> z[3] + 32 Pi <a, n>  z[3]
>     ------------------------------------------------------ +
                                8 Pi <n, a'>

      8 Pi <n, a'> z[4]
>     ----------------- }
        8 Pi <n, a'>

In[52]:= sol = Join[sol,sol0]

Out[52]= {z[1] ->

                                                      2
      -(f <n, a'>) - 16 Pi <a, a> z[3] + 32 Pi <a, n>  z[3]
>     ------------------------------------------------------ +
                                8 Pi <n, a'>

      8 Pi <n, a'> z[4]
>     -----------------
        8 Pi <n, a'>

                -3 e
>     , z[7] -> -----}
                32 Pi

In[53]:= gl = gl /. sol0 //Expand

                                                       2
         2 <a, n> a z[3]   2 <a, a> a' z[3]   4 <a, n>  a' z[3]
Out[53]= --------------- + ---------------- - -----------------
                r             r <n, a'>          r <n, a'>

In[54]:= gl = Collect[gl,a']
```

```
                                                         2
         2 <a, n> a z[3]       2 <a, a> z[3]   4 <a, n>  z[3]
Out[54]= --------------- + a' (------------- - --------------)
                r                r <n, a'>       r <n, a'>

In[55]:= part = Last[gl]

                                      2
             2 <a, a> z[3]   4 <a, n>  z[3]
Out[55]= a' (------------- - --------------)
               r <n, a'>       r <n, a'>

In[56]:= sol0 = Solve[part == 0, z[3]][[1]]

Out[56]= {z[3] -> 0}

In[57]:= sol = sol /. sol0

                  -(f <n, a'>) + 8 Pi <n, a'> z[4]          -3 e
Out[57]= {z[1] -> --------------------------------, z[7] -> -----}
                            8 Pi <n, a'>                    32 Pi

In[58]:= sol = Join[sol,sol0]

                  -(f <n, a'>) + 8 Pi <n, a'> z[4]          -3 e
Out[58]= {z[1] -> --------------------------------, z[7] -> -----,
                            8 Pi <n, a'>                    32 Pi

>    z[3] -> 0}

In[59]:= gl = gl /. sol0 //Expand

Out[59]= 0
```

The determinitation of z[1],...,z[7] is finished. Finally, we calculate A_0, d_0 and b_0 from the obtained values of z[1],...,z[7].

```
In[61]:= A[0] = A[0] /. sol //Expand

         -(f r a')   3 e r <a, n> (a x n)
Out[61]= --------- + -------------------- + r (n x a') z[2] +
           8 Pi             32 Pi

>    r <n, a'> n z[4] + r a' z[4] + r <a, a> n z[5] + r <a, n> a z[6]

In[62]:= d[0] = rot[A[0]] //Expand
```

Table 3. The number of terms in the ansatz of A_n.

step n	number of terms
-1	3
0	7
1	17
2	37
3	78
4	157
5	308
6	584

```
                                                       2
         3 e <a, n> a   3 e <a, n> n   3 e <a, n>  n   f (n x a')
Out[62]= ------------ - ------------ + ------------- - ---------- -
            16 Pi          32 Pi           32 Pi          8 Pi

>    2 a' z[2]

In[63]:= b[0] = d[0] /. symrule

                                                       2
         3 f <a, n> a   3 f <a, a> n   3 f <a, n>  n   e (n x a')
Out[63]= ------------ - ------------ + ------------- + ---------- -
            16 Pi          32 Pi           32 Pi          8 Pi


>    2 a' z[2]
```

We have determined the special solutions of (7.1) considered in this section up to n = 6, to support the investigations of [1]. For this purpose we have written two extension packages MAGNET 1.0 and MAGNET 2.0 of VECTAN which self-acting generate the terms of the ansatzes for the A_n and determine the unknown constants $z_{n,k}$. Table 3 gives the number of terms in the ansatzes of the A_n. The rules for the construction of these terms we have not explained for reasons of shortness in this program description.

8. References of the commands of VECTAN

```
BacCab::usage =
"BacCab[expr,x,y,z] applies the bac-cab-rule VP[x,VP[y,z]] -> SP[x,z]y -
SP[x,y]z to the expression expr by means of ReplaceAll."

DefineVector::usage =
"DefineVector[v] declares the expression v a vector
field. DefineVector can be used to a list of expressions too
(DefineVector[{v1,v2,...}])."

div::usage =
```

```
"div[vect] denotes the divergence of the vector field vect."

divValue::usage =
"A value wert can be assigned to the operation div[vect] on the vector
field vect by means of a statement vect/: divValue[vect] := wert . See
the comments in def.m and the explanations in vectan.ps."

divValue::muster =
"VECTAN identifies only the pattern divValue[vect]."

dot::usage =
"dot[x] denotes the derivative of x according to a variable t, which
differs from the spatial variables x1, x2, x3. If x is not a vector
field, then there holds true dot[x] = Derivative[1][x]. If x is a
vector field x = Vect[x], then there is valid dot[x] = Vect[x']."

dotValue::usage =
"A value wert can be assigned to the operation dot[x] on the symbol x
by means of a statement x/: dotValue[x] := wert. Here x may be a
vector field as well as a function. See the comments in def.m and the
explanations in vectan.ps."

dotValue::muster =
"VECTAN identifies only the patterns dotValue[vect] and dotValue[func]."

grad::usage =
"grad[f] denotes the gradient of the function f. grad[vect1,vect2]
denotes the derivative of the vector field vect2 in the direction of
the vector field vect1, i.e. (<vect1,grad>)[vect2] ."

gradValue::usage =
"A value wert can be assigned to the operation grad[f] on the function f
by means of a statement f/: gradValue[f] := wert. Furthermore, a value
wert can be assigned to the operation grad[vect1,vect2] on the vector
fields vect1 and vect2 by means of one of the statements
vect1/: gradValue[vect1,vect2] := wert or
vect2/: gradValue[vect1,vect2] := wert. See the comments in def.m and
the explanations in vectan.ps."

gradValue::muster =
"VECTAN identifies the following patterns:
gradValue[func], gradValue[vec1,vec2],
gradValue[vec1,grad[func]], gradValue[vec1,rot[vec2]],
gradValue[vec1,VP[vec2,vec3]], gradValue[vec1,grad[vec2,vec3]],
gradValue[grad[func],vec1], gradValue[rot[vec1],vec2],
gradValue[VP[vec1,vec2],vec3], gradValue[grad[vec1,vec2],vec3]."

graddotValue::usage =
```

```
"A value wert can be assigned to the operation
grad[m-th dot-derivative of  vect1, n-th dot-derivative of vect2]
by means of a statement
vect1/: graddotValue[vect1,vect2,m,n] := wert. A statement
vect2/: graddotValue[vect1,vect2,m,n] := wert has the same effect.
Furthermore the numbers m, n are allowed to be zero. A value for a
pattern grad[x_, Vect[Derivative[n][vect2]]] can be defined by
vect2/: graddotValue[x_,vect2,0,n] := wert, where wert may be
depending on x. In the same way, a value for a pattern
grad[Vect[Derivative[m][vect1]],x_] can be established by
vect1/: graddotValue[vect1,x_,m,0] := wert. See the comments in def.m
and the explanations in vectan.ps."

graddotValue::muster =
"VECTAN identifies the following patterns:
graddotValue[vect1,vect2,m,n] (m,n >= 1),
graddotValue[vect1,vect2,m,0] (m >= 1),
graddotValue[vect1,vect2,0,n] (n>= 1).
The non-differenciated vectors of the last two patterns may even be
expressions with the heads grad, rot, VP."

Mu::usage =
"Mu[symbol] prints the message symbol::muster, if such a message is
defined in VECTAN. The symbols divValue, dotValue, gradValue,
graddotValue, rotValue, SPValue, VPValue possess such messages."

rot::usage =
"rot[vect] denotes the curl of the vector field vect."

rotValue::usage =
"A value wert can be assigned to the operation rot[vect] on the vector
field vect by means of a statement vect/: rotValue[vect] := wert . See
the comments in def.m and the explanations in vectan.ps."

rotValue::muster =
"VECTAN identifies only the pattern rotValue[vect]."

SP::usage =
"SP[vect1,vect2] denotes the scalar product of the vector fields vect1,
vect2."

SPValue::usage =
"A value wert can be assigned to the expression SP[vect1,vect2]
by means of a statement vect1/: SPValue[vect1,vect2] := wert or vect2/:
SPValue[vect1,vect2] := wert . See the
comments in def.m and the explanations in vectan.ps."

SPValue::muster =
```

```
"VECTAN identifies the following patterns:
SPValue[vec1,vec2], SPValue[grad[f],vec],
SPValue[rot[vec1],vec2]], SPValue[grad[vec1,vec2],vec3],
SPValue[vec1,VP[vec2,vec3]]."

VASave::usage =
"VASave[filename] stores all definitions which are contained in the
current context in the file filename. Thus VASave stores all
definitions generated in a VECTAN session if and only if all these
definitions belong to the current context. Therefore, the current
context should not be changed in the course of a VECTAN session."

Vect::usage =
"Vect is the head of an expression which VECTAN consideres a vector
field. The FullForm of a vector field vect is Vect[vect]."

VectQ::usage =
"VectQ[x] yields True if x possesses one of the heads Vect, grad, rot,
VP. Otherwise, Vect[x] yields False."

VP::usage =
"VP[vect1,vect2] denotes the vector product of the vector fields vect1,
vect2."

VPValue::usage =
"A value wert can be assigned to the expression VP[vect1,vect2]
by means of a statement vect1/: VPValue[vect1,vect2] := wert or vect2/:
VPValue[vect1,vect2] := wert . See the
comments in def.m and the explanations in vectan.ps."

VPValue::muster =
"VECTAN identifies the following patterns:
VPValue[vec1,vec2], VPValue[grad[f],vec],
VPValue[rot[vec1],vec2], VPValue[grad[vec1,vec2],vec3]."

$BacCab::usage =
"If $BacCab = True, then the rule VP[a,VP[b,c]] = SP[a,c]b - SP[a,b]c
is used automatically. If $BacCab = False, then this rule can be
applied only by means of the command BacCab[expr,a,b,c]."

$KillSpat::usage =
"If $KillSpat = True, then expressions in the form (scalar
triple product)*vector are automatically converted into expressions in
the form (scalar product)*vector, where 'scalar product' is not a
scalar triple product. 'vector' may be an expression u with VectQ[u] =
True."

$SpatRule::usage =
```

```
"If $SpatRule = True, then products of scalar triple products like
SP[a,VP[b,c]]*SP[d,VP[e,f]] are automatically converted into sums of
products of scalar products, where the scalar products do not contain
a vector product."

$VPChar::usage =
"The variable $VPChar contains a string that is used to print the sign
of the vector product."
```

9. Table of contents

References

[1] H.-P. Gittel, J. Kijowski, and E. Zeidler. The relativistic dynamics of the combined particle-field system in nonlinear renormalized electrodynamics. *J. Math. Phys.*, 1996. will be sended in.