```
kill(all)$
```

maxima:prog:appendix "vector product evaluation with infix("##") operator",version 1.0 a
peter.vlasschaert@gmail.com,20/10/2023

sign ,signum :

→ Function: sign (expr):

Attempts to determine the sign of expr on the basis of the facts in the current data base.
It returns one of the following answers: pos (positive), neg (negative), zero, pz (positive or zero),
nz (negative or zero), pn (positive or negative), or pnz (positive, negative, or zero, i.e. nothing known).

→ Function: signum (x):

For either real or complex numbers x, the signum function returns 0 if x is zero; for a nonzero numeric input x, the signum function returns x/abs(x).

For non-numeric inputs, Maxima attempts to determine the sign of the input.
When the sign is negative, zero, or positive, signum returns -1,0, 1, respectively.

For all other values for the sign, signum a simplified but equivalent form. The simplifications include reflection (signum(-x) gives -signum(x)) and multiplicative identity (signum(x*y) gives signum(x) * signum(y)).

Given :We want to remove elements from list.

```
p1:[−(e[1]·x·y),−(c·d·e[3])];
```

$$\left[-(e_1\,x\,y),-(e_3\,c\,d)\right]$$

```
p1a:listofvars(p1);
```

$$\left[e_1,x,y,e_3,c,d\right]$$

rem : call it → elt = element

How to remove:[e[1],e[2],e[3]] from "p1a"

```
remove_elements: [e[1], e[2], e[3]];
```

$$\left[e_1,e_2,e_3\right]$$

→ makelist

```
L1: p1a;
L1: makelist(if not member(elt, remove_elements) then elt, elt, L1);
L1:delete(false,L1);
```

$$\left[e_1,x,y,e_3,c,d\right] \qquad [\text{false},x,y,\text{false},c,d] \qquad [x,y,c,d]$$

rem : How to use makelist with some condition, " member or not member".
makelist(condition on elt,elt,L1)

→ for loop (in) , used also in python

```
L2:p1a;
```

$$\left[e_1,x,y,e_3,c,d\right]$$

```
for elt in remove_elements do (
   L2: delete(elt, L2)
)$
print(L2)$
```

$$[x,y,c,d]$$

→ difference , use set :{} ,not a list:[]

```
L3:p1a;
```

$$\left[e_1,x,y,e_3,c,d\right]$$

```
L3: setdifference(setify(L3), setify(remove_elements));
L3:listify(L3);
```

$$\{c,d,x,y\} \qquad [c,d,x,y]$$

set are all elements "only" ones.
list → set : setify ,set → list : listify.

find : list contain only .[e[i],e[j]], i,j = {1,2,3}

→ list → sublist

```
L4: p1a;

/* Filtering function */
is_desired_form(elt) := is(atom(elt) = false and op(elt) = e);/*e[i] this e*/

L4: sublist(L4, lambda([elt], is_desired_form(elt)));/*sublist from p1a*/
```

$$\left[e_1,x,y,e_3,c,d\right] \qquad \text{is\_desired\_form}(elt):=\text{is}(\text{atom}(elt)=\text{false}\ \textit{and}\ \text{op}(elt)=e) \qquad \left[e_1,e_3\right]$$

rem : "is" check condition

```
is(infix("##")="##");
```

true

```
is(2>5 and 7<110);/* both condition must be true , result:true*/
```

false

```
m1:map(atom,p1a);/* atom , command on all of elements p1a*/
m2:map(atom,flatten([p1a,"+",2,"home",r^2,∞,∫,diff,Limit,plot2d]));
```

[false, true, true, false, true, true] [false, true, true, false, true, true, true, true, true, false, true, true, true, true, true]

```
print(" op(e[i]) = ",op(e[i]))$
```

op(e[i]) = e

rem : " lambda " , f(x) = sin(x)

```
f:lambda([x], sin(x))$
```

syntax :→ func:lambda(list,condition)
also : f(x) := sin(x) ,same as f

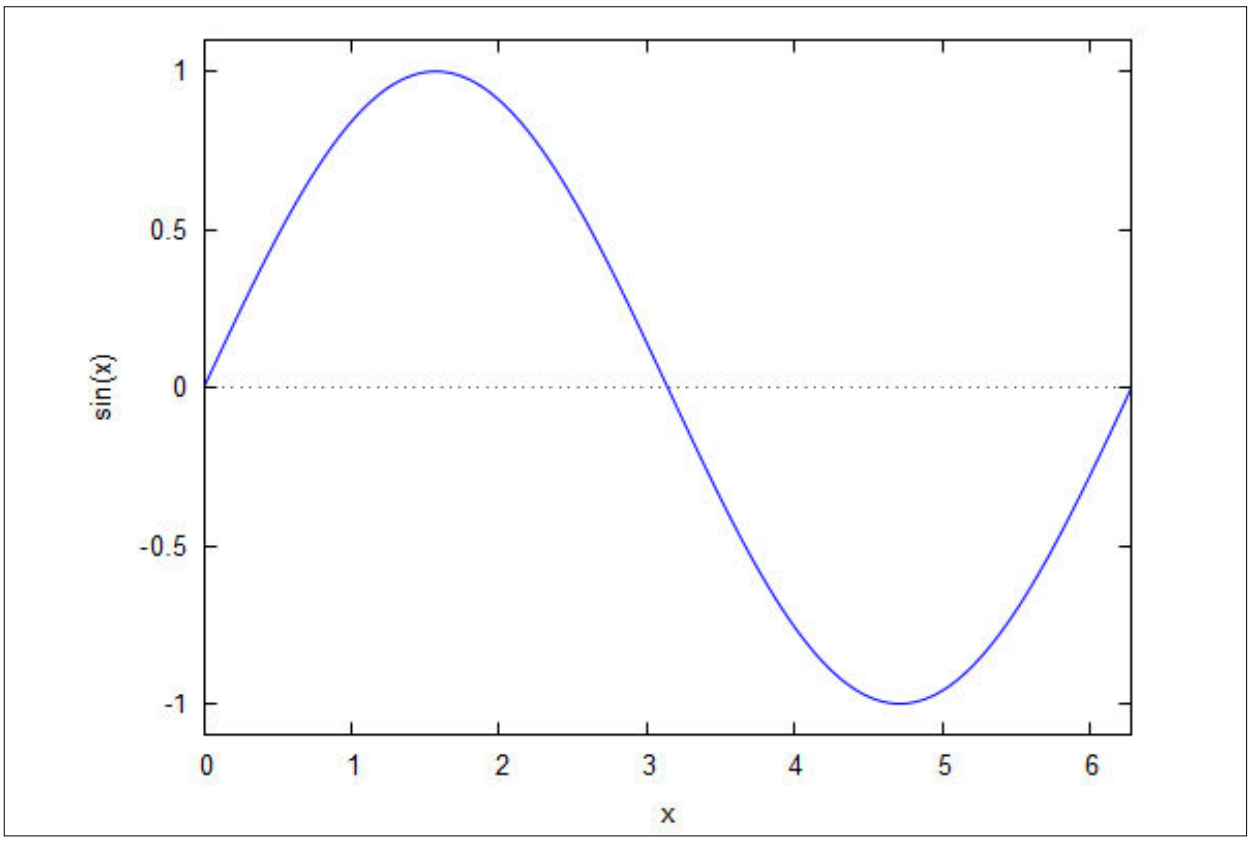
```
m3:print( " x → f(x) = ", f(%pi/2))$
```

x → f(x) = 1

rem : "lambda,sublist" ,f(x) = sin(x) =0, "condition"
---

four values : given see graph

```
m4:[0,%pi/2,%pi,2·%pi]$
wxplot2d([sin(x)], [x,0,2·%pi])$
```



plot2d: sin(x) , inline plot

```
m4:sublist(m4,lambda([x], sin(x)= 0));
```

[0, π, 2 π]

explain :
======

We define a function is_desired_form that checks whether a given element is of the form e[something].
We then use sublist with a lambda function to filter out the elements of L that satisfy the desired form.

→ do loop

list →

```
L5: p1a;
```

$[e_1, x, y, e_3, c, d]$

→ atom : ?

```
map(atom,L5);/* false :index variables*/
```

[false, true, true, false, true, true]

```
L5: block(
  [output],
  output: [],
  for elt in L5 do (
     if atom(elt) = false and part(elt, 0) = e then (
        output: append(output, [elt])
     )
  ),
  return(output)
)$
```

sublist →

```
print ("sublist =output, [e[1],x,y,e[3],c,d] → ",L5)$
```

sublist =output, [e[1],x,y,e[3],c,d] → $[e_1, e_3]$

other condition : to filter out.[e[1],e[3]]
==========

```
L6: p1a;
res: block(
   [output, unwanted],
   output: [],
   unwanted: [e[1], e[2]], /* List of elements to filter out */
   for elt in L6 do (
      /* Check if the element is not in the unwanted list */
      if not member(elt, unwanted) then (
         output: append(output, [elt])
      )
   ),
   return(output)
);
```

$[e_1, x, y, e_3, c, d]$ $[x, y, e_3, c, d]$

```
L7: p1a;
L7: block(
   [output, wanted],
   output: [],
   wanted: [e[1], e[3]], /* List of elements we want to extract */
   for elt in L7 do (
      /* Check if the element is in the wanted list */
      if member(elt, wanted) then (
         output: append(output, [elt])
      )
   ),
   return(output)
);
```

$[e_1, x, y, e_3, c, d]$ $[e_1, e_3]$

vector product :e[1]##e[3]

1e) representation : vector product
---
unit vectors : cartesian coordinates.
---
e[1] , x-direction
e[2] , y-direction
e[3] , z-direction

==========
rule : e[1]##e[2] = e[3] , right handed system => e[2]##e[1] = -e[3]
e[i]##e[j] = if i=j then 0 else result
cyclic permutation : 1→2→3→1
==========0

solved : problem for unit vector = "A","B" ?

```
k1:infix("##");
```

##

→ find : "A"

```
k2:"e[1]##e[3]" = "A"; /*find */
```

e[1]##e[3] = A

```
k3:split(lhs(k2),"##");
```

[e[1] , e[3]]

```
k4:[parse_string(k3[1]),parse_string(k3[2])];
```

$[e_1, e_3]$

```
k5:flatten([args(k4[1]),args(k4[2])]);
```

[1 , 3]

→ find : "B"

```
k6:"e[2]##e[1]"="B";
```

e[2]##e[1] = B

make function : product rule.

method 1 : simple

```
vector_product(u, v) :=
  block(
    [i, j],
    /* Extract the indices from the vectors */
    i: args(u)[1],
    j : args(v)[1],

    /* Check for vector product rules */
    if (i = j) then return(0) /* If the vectors are the same, return 0 */
    else if (i = 1 and j = 2) then return(e[3])
    else if (i = 2 and j = 3) then return(e[1])
    else if (i = 3 and j = 1) then return(e[2])
    else return(−1 · vector_product(v, u)) /* Use antisymmetry property */
  )$

grind(vector_product)$
```

```
vector_product(u,v):=block([i,j],i:args(u)[1],j:args(v)[1],
                if i = j then return(0)
                    else (if i = 1 and j = 2 then return(e[3])
                              else (if i = 2 and j = 3 then return(e[1])
                                        else (if i = 3 and j = 1
                                                  then return(e[2])
                                                  else return(
                                                  -1*vector_product(v,u))))))$
```

solved problem :

```
/* Test the function */
A: A=vector_product(e[1], e[3]) ; /* Expected  -e[2] */
B: B=vector_product(e[2], e[1])  ; /* Expected -e[3] */
```

$A=-e_2$ $B=-e_3$

method 2:kronecker delta function

---

info : kronecker delta

```
epsilon(i, j , k) :=
  if (i = j or j = k or k = i) then 0
  else if (i = 1 and j = 2 and k = 3) then 1
  else if (i = 2 and j = 3 and k = 1) then 1
  else if (i = 3 and j = 1 and k = 2) then 1
  else −1;

vector_product_kron(u, v) :=
  block(
    [result, i, A, B],
    A: args(u)[1],
    B: args(v)[1],
    result: sum(epsilon(i, A, B) · e[i], i, 1, 3)
  )$

/* Test the function */
A1:(vector_product_kron(e[1], e[3])); /* Expected - e[2] */
B1 :vector_product_kron(e[2], e[1]); /* Expected - e[3] */
```

$\text{epsilon}(i,j,k):=\text{if } i=j \text{ or } j=k \text{ or } k=i \text{ then } 0 \text{ else if } i=1 \text{ and } j=2 \text{ and } k=3 \text{ then } 1 \text{ else if } i=2 \text{ and } j=3 \text{ and } k=1 \text{ then } 1 \text{ else if } i=3 \text{ and } j=1 \text{ and } k=2 \text{ then } 1 \text{ else } -1$ $-e_2$ $-e_3$

```
print("e[1]##e[3] =",string(A1))$
print("e[2]##e[1] =",string(B1))$
```

e[1]##e[3] = −e[2]

e[2]##e[1] = −e[3]

rem :You can copy and paste the above code into TeXStudio (or any other LaTeX editor),
then compile it to produce a PDF with the given explanation.
==========
→TexStudion : free software for scientific papers : *.tex to *.pdf
==========
1e) install Texstudio
2e) New file
3e) Compile
==========

explain : code: latex , *.tex

```
\documentclass{article}
\usepackage{amsmath, amssymb}

\begin{document}

\section*{Cross Product using Kronecker Delta and Levi-Civita Symbol}

In three-dimensional Cartesian coordinates, the cross product of two vectors, \(\mathbf{A}\) and \(\mathbf{B}\), is given by:

\[
(\mathbf{A} \times \mathbf{B})_i = \sum_{j=1}^{3} \sum_{k=1}^{3} \varepsilon_{ijk} A_j B_k
\]

where \(i, j,\) and \(k\) are the Cartesian indices (1 for x, 2 for y, and 3 for z), \(\mathbf{A} = (A_1, A_2, A_3)\) and \(\mathbf{B} = (B_1, B_2, B_3)
the components of vectors \(\mathbf{A}\) and \(\mathbf{B}\) respectively, and \(\varepsilon_{ijk}\) is the Levi-Civita symbol.

The Levi-Civita symbol, \(\varepsilon_{ijk}\), is defined as:

\[
\varepsilon_{ijk} = \left\{
\begin{array}{ll}
+1 & \text{if } (i,j,k) \text{ is a cyclic permutation of } (1,2,3) \\
-1 & \text{if } (i,j,k) \text{ is an anti-cyclic permutation} \\
0 & \text{otherwise}
\end{array}
\right.
\]

In our Maxima code:

\begin{enumerate}
    \item The function \texttt{epsilon(i, j, k)} captures this definition of the Levi-Civita symbol.
    \item The \texttt{vector\_product\_kron(u, v)} function computes the cross product based on the above formula. Within this function:
    \begin{itemize}
        \item The components of the vectors \texttt{u} and \texttt{v} are extracted using \texttt{args(u)[1]} and \texttt{args(v)[1]}, respectively.
        \item The \texttt{sum} function is employed to handle the double summation over \(j\) and \(k\).
        \item The product \(\varepsilon_{ijk} A_j B_k\) is computed for each \(i\), yielding the components of the resultant cross product.
    \end{itemize}
    \item The test cases check the function's output against known results of cross products for unit vectors.


\end{enumerate}

In the context of this discussion, the Kronecker delta hasn't been explicitly used, but it's worth noting that the Kronecker delta, \(\delta_{ij}\), is
as:

\[
\delta_{ij} = \left\{
\begin{array}{ll}
1 & \text{if } i = j \\
0 & \text{otherwise}
\end{array}
\right.
\]

The Kronecker delta effectively captures the notion that two identical components (same direction) yield a scalar product of one, whereas different
components (different directions) yield zero. In the case of the cross product, it's the Levi-Civita symbol that plays the key role.

\end{document}
```

---

info :

```
/* Define the new infix operator '##' with the highest precedence */
infix("##", 80)$

/* Levi-Civita symbol */
epsilon(i, j , k) :=
   if (i = j  or j  = k or k = i) then 0
   else if (i = 1 and j  = 2 and k = 3) then 1
   else if (i = 2 and j  = 3 and k = 1) then 1
   else if (i = 3 and j  = 1 and k = 2) then 1
   else −1$

/* Vector product function */
vector_products(a, u, b, v) :=
   block([i, j , k, result],
      i: first(u),
      j : first(v),
      result: 0,
      for k:1 thru 3 do (
         result: result + epsilon(i, j , k)·e[k]
      ),
      a · b · result
   )$

/* Test expression */
expr: vector_products(a·b, e[1], x·y, e[1]) −
    vector_products(c·d, e[1], m·n, e[2]) +
    vector_products(e·f, e[2], p·q, e[2]) +
    vector_products(g·h, e[3], r·s, e[3])$

print("vector_products(a*b, e[1], x*y, e[1]) -
    vector_products(c*d, e[1], m*n, e[2]) +
    vector_products(e*f, e[2], p*q, e[2]) +
    vector_products(g*h, e[3], r*s, e[3])",expr)$
```

*vector_products(a*b, e[1], x*y, e[1]) − vector_products(c*d, e[1], m*n, e[2]) + vector_products(e*f, e[2], p*q, e[2]) + vector_products(g*h, e[3], r*s, e[3])* $-(e_3\ c\ d\ m\ n)$

final result :

vector_products(a*b, e[1], x*y, e[1]) -
vector_products(c*d, e[1], m*n, e[2]) +
vector_products(e*f, e[2], p*q, e[2]) +
vector_products(g*h, e[3], r*s, e[3]) =

expr;

$$-(e_3\ c\ d\ m\ n)$$

==========

```
(%i6)   /* Define the new infix operator '##' with the highest precedence */
    infix("##", 80);

    /* Levi-Civita symbol */
    epsilon(i, j, k) :=
        if (i = j or j = k or k = i) then 0
        else if (i = 1 and j = 2 and k = 3) then 1
        else if (i = 2 and j = 3 and k = 1) then 1
        else if (i = 3 and j = 1 and k = 2) then 1
        else -1;

    /* Define the behavior for our custom operator '##' */
    vector_product(a, i, b, j) :=
        block([result, k],
            result: 0,
            for k:1 thru 3 do (
                result: result + epsilon(i, j, k)*e[k]
            ),
            return(a * b * result)
        );

    /* Override the behavior of the '##' operator */
    myop(u, v) :=
        block([a, i, b, j],
            a: coeff(u, e[1]), i: 1,
            if a = 0 then (a: coeff(u, e[2]), i: 2),
            if a = 0 then (a: coeff(u, e[3]), i: 3),

            b: coeff(v, e[1]), j: 1,
            if b = 0 then (b: coeff(v, e[2]), j: 2),
            if b = 0 then (b: coeff(v, e[3]), j: 3),

            return(vector_product(a, i, b, j))
        );

    /* Test expression */
    expr: myop(a*b*e[1], x*y*e[1]) - myop(c*d*e[1], m*n*e[2]) + myop(e*f*e[2], p*q*e[2]) + myop(g*h*e[3], r*s*e[3]);

    print(expr);

(%o1)   "##"

(%o2)   epsilon(i,j,k):=if i=j or j=k or k=i then 0 else if i=1 and j=2 and k=3 then 1 else if i=2 and j=3 and k=1 then 1 else if i=3 and j=1 and k=2 th
else -1

(%o3)   vector_product(a,i,b,j):=block([result,k],result:0,for k thru 3 do result:result+epsilon(i,j,k)*e[k],return(a*b*result))
(%o4)   myop(u,v):=block([a,i,b,j],a:coeff(u,e[1]),i:1,if a=0 then (a:coeff(u,e[2]),i:2) ,if a=0 then (a:coeff(u,e[3]),i:3) ,b:coeff(v,e[1]),j:1,if b=0
(b:coeff(v,e[2]),j:2) ,if b=0 then (b:coeff(v,e[3]),j:3) ,return(vector_product(a,i,b,j)))
(%o5)   -(e[3]*c*d*m*n)
-(e[3]*c*d*m*n)" "

(%o6)   -(e[3]*c*d*m*n)
```

```
/* Define a new infix operator '##' with precedence */
infix("##", 67);  /* 67 is chosen so that it lies between * (70) and + (65) */

/* Implement the behavior for our custom operator */
matchdeclare(a, true, b, true);  /* This allows for any match for 'a' and 'b' */
myop(a##b) := a * b - 3;  /* Example implementation: product of operands minus 3 */

/* Test our new operator */
expr1 := x##y + z; /* Since ## has precedence over +, this evaluates as (x##y) + z */
expr2 := x##y * z; /* Since * has precedence over ##, this evaluates as (x##y) * z */

/* Display the results */
print(expr1);  /* Expected output: x*y - 3 + z */
print(expr2);  /* Expected output: z*(x*y - 3) */
```

==========

appendix :

a) remove operator : ##

```
fexpr: "a*b*e[1]##x*y*e[1] - c*d*e[1]##m*n*e[2] + e*f*e[2]##p*q*e[2] + g*h*e[3]##r*s*e[3]";
fexpr1:split(fexpr,"##");/* need : combine*/
pe1:fexpr1[2];
pe2:makelist(fexpr1[i],i,2,length(fexpr1)-1);
```

$a*b*e[1]\#\#x*y*e[1] - c*d*e[1]\#\#m*n*e[2] + e*f*e[2]\#\#p*q*e[2] + g*h*e[3]\#\#r*s*e[3]$

$[a*b*e[1], x*y*e[1] - c*d*e[1], m*n*e[2] + e*f*e[2], p*q*e[2] + g*h*e[3], r*s*e[3]]$ $\quad x*y*e[1] - c*d*e[1]$

$[x*y*e[1] - c*d*e[1], m*n*e[2] + e*f*e[2], p*q*e[2] + g*h*e[3]]$

b) split terms with "+" or "-" :

```
split_terms(s) :=
   block(
      str_modified: ssubst(" ,", "+", s),
      str_modified: ssubst(",-", "- ", str_modified),
      str_list: split(str_modified, ","),
      /* Ensure str_list is a list of strings */
      makelist(parse_string(str_list[i]), i, 1, length(str_list))
   )$

/* Test the helper function */
test_output: split_terms("x*y*e[1] - c*d*e[1]"); /* Expected ["x*y*e[1]", "-c*d*e[1]"] */

/* Split each expression in the list */
terms_list: flatten(makelist(split_terms(x), x, ["x*y*e[1] - c*d*e[1]", "m*n*e[2] + e*f*e[2]", "p*q*e[2] + g*h*e[3]"]));
terms_list1: flatten(makelist(split_terms(x), x, pe2));
```

$$[e_1\,x\,y,-(e_1\,c\,d)]\qquad[e_1\,x\,y,-(e_1\,c\,d),e_2\,m\,n,e_2\,e\,f,e_2\,p\,q,e_3\,g\,h]\qquad[e_1\,x\,y,-(e_1\,c\,d),e_2\,m\,n,e_2\,e\,f,e_2\,p\,q,e_3\,g\,h]$$

c) combine:

```
pc1:terms_list1;
pc2:flatten([eval_string(first(fexpr1)),pc1,eval_string(last(fexpr1))]);
pc3:length(pc2)/2;
pc4:makelist([pc2[2·i−1],pc2[2·i]],i,1,pc3);
```

$$[e_1\,x\,y,-(e_1\,c\,d),e_2\,m\,n,e_2\,e\,f,e_2\,p\,q,e_3\,g\,h]\qquad[e_1\,a\,b,e_1\,x\,y,-(e_1\,c\,d),e_2\,m\,n,e_2\,e\,f,e_2\,p\,q,e_3\,g\,h,e_3\,r\,s]\qquad 4$$

$$[[e_1\,a\,b,e_1\,x\,y],[-(e_1\,c\,d),e_2\,m\,n],[e_2\,e\,f,e_2\,p\,q],[e_3\,g\,h,e_3\,r\,s]]$$

d) representation : by example vectorproduct , use rules.

```
pd1:print(fexpr,"→" ,pc4)$
```

a*b*e[1]##x*y*e[1] − c*d*e[1]##m*n*e[2] + e*f*e[2]##p*q*e[2] + g*h*e[3]##r*s*e[3] → $[[e_1\,a\,b,e_1\,x\,y],[-(e_1\,c\,d),e_2\,m\,n],[e_2\,e\,f,e_2\,p\,q],[e_3\,g\,h,e_3\,r\,s]]$

rem :

```
[op(e[1]),args(e[1])];
```

$$[e,[1]]$$

---

example 1: (vec)tor (prod)uct
---

use : pc4

```
kill(all)$
original_list: [[e[1]·a·b,e[1]·x·y],[−(e[1]·c·d),e[2]·m·n],[e[2]·e·f,e[2]·p·q],[e[3]·g·h,e[3]·r·s]];

/* Function to remove e[i] from the term, i=1,2,3 */
remove_ei(term) :=
   block(
      term_no_e1: subst(1, e[1], term),
      term_no_e2: subst(1, e[2], term_no_e1),
      term_no_e3: subst(1, e[3], term_no_e2)
   )$

/* Apply the function to each term in the list */
modified_list: makelist(makelist(remove_ei(original_list[i][j]), j, 1, 2), i, 1, length(original_list))$


pm0:modified_list;

pm1:listofvars(modified_list);
```

$$[[e_1\,a\,b,e_1\,x\,y],[-(e_1\,c\,d),e_2\,m\,n],[e_2\,e\,f,e_2\,p\,q],[e_3\,g\,h,e_3\,r\,s]]\qquad[[a\,b,x\,y],[-(c\,d),m\,n],[e\,f,p\,q],[g\,h,r\,s]]$$

$$[a,b,x,y,c,d,m,n,e,f,p,q,g,h,r,s]$$

```
pm2a:product(part(pm0[1],j),j,1,length(pm0[1]));
```

$$a\,b\,x\,y$$

→ coef :

```
pm2:makelist(product(part(pm0[i],j),j,1,length(pm0[i])),i,1,length(pm0));
```

$$[a\,b\,x\,y,-(c\,d\,m\,n),e\,f\,p\,q,g\,h\,r\,s]$$

→ sign:

```
pm3a:makelist(sign(pm2[i]),i,1,length(pm2));/*reason see before definition*/
```

$$[pnz,pnz,pnz,pnz]$$

```
pm3b:makelist(pm1[i]=1,i,1,length(pm1));
```

$$[a=1,b=1,x=1,y=1,c=1,d=1,m=1,n=1,e=1,f=1,p=1,q=1,g=1,h=1,r=1,s=1]$$

```
pm3c:subst(pm3b,pm0);
```

$$[[1,1],[-1,1],[1,1],[1,1]]$$

```
pm3d:makelist(pm3c[i][1]·pm3c[i][2],i,1,length(pm3c));
```

$$[1,-1,1,1]$$

→[e[i],e[j]]

```
pm4:original_list/pm0;
```

$$\left[\left[e_1,e_1\right],\left[e_1,e_2\right],\left[e_2,e_2\right],\left[e_3,e_3\right]\right]$$

use .pm4[2]

```
pm4a:flatten([args(pm4[2][1]),args(pm4[2][2])]);
```

$$[1,2]$$

```
pm4b:makelist(flatten([args(pm4[i][1]),args(pm4[i][2])]),i,1,length(pm4));
```

$$[[1,1],[1,2],[2,2],[3,3]]$$

vector product : rule

```
vec_prod_rules(i, j) :=
   if i=j  then 0
   else if i=1 and j =2 then e[3]
   else if i=2 and j =1 then −e[3]
   else if i=1 and j =3 then −e[2]
   else if i=3 and j =1 then e[2]
   else if i=2 and j =3 then e[1]
   else if i=3 and j =2 then −e[1]
   else 0$
```

use : coef pm2,vector rules:pm4b

```
m5:makelist(vec_prod_rules(pm4b[i][1],pm4b[i][2])·pm2[i],i,1,length(pm4));
```

$$\left[0,-(e_3\ c\ d\ m\ n),0,0\right]$$

final result :vector product.

```
m5a:sum(m5[i],i,1,length(pm4));
```

$$-(e_3\ c\ d\ m\ n)$$

final :solution presentation, ## represent : vector product

```
mm:print( "a*b*e[1]##x*y*e[1] - c*d*e[1]##m*n*e[2] + e*f*e[2]##p*q*e[2] + g*h*e[3]##r*s*e[3]","=",m5a)$
```

$$a*b*e[1]\#\#x*y*e[1] - c*d*e[1]\#\#m*n*e[2] + e*f*e[2]\#\#p*q*e[2] + g*h*e[3]\#\#r*s*e[3] = -(e_3\ c\ d\ m\ n)$$

example 2:(sca)lar (prod)uct

→ use : pm4,pm4b , unit vectors.

```
qm1:pm4;
qm2:pm4b;
```

$$\left[\left[e_1,e_1\right],\left[e_1,e_2\right],\left[e_2,e_2\right],\left[e_3,e_3\right]\right] \qquad [[1,1],[1,2],[2,2],[3,3]]$$

→ use : pm2

```
qm3:pm2;
```

$$[a\ b\ x\ y,-(c\ d\ m\ n),e\ f\ p\ q,g\ h\ r\ s]$$

(sca)lar (prod)uct : rule

```
sca_prod_rules(i, j) := if i=j  then 1 else 0 $
```

→ computation

```
qm4:makelist(sca_prod_rules(qm2[i][1],qm2[i][2])·qm3[i],i,1,length(qm1));
```

$$[a\ b\ x\ y,0,e\ f\ p\ q,g\ h\ r\ s]$$

final result : scalar product

```
qm4a:sum(qm4[i],i,1,length(qm1));/* result is value ,here variables:symbolic*/
```

$$a\ b\ x\ y+g\ h\ r\ s+e\ f\ p\ q$$

final :solution presentation, ## represent : scalar product

```
nn:print( "a*b*e[1]##x*y*e[1] - c*d*e[1]##m*n*e[2] + e*f*e[2]##p*q*e[2] + g*h*e[3]##r*s*e[3]","=",qm4a)$
```

$$a*b*e[1]\#\#x*y*e[1] - c*d*e[1]\#\#m*n*e[2] + e*f*e[2]\#\#p*q*e[2] + g*h*e[3]\#\#r*s*e[3] = a\ b\ x\ y+g\ h\ r\ s+e\ f\ p\ q$$

==========