Higher order perturbations

```
%hide
def quick(l):
    l = list(l)
    exchanges = []
    n = len(l)
    i = 0
    while i < n:
        if (i>0) and (l[i]<l[i-1]):
            new_place = min([u for u in range(i) if l[u]>l[i]
])
            tt = l[new_place]
            l[new_place] = l[i]
            l[i] = tt
            exchanges.append( ( new_place,i ) )
        else:
            i += 1
    return [l,exchanges]
```

```
class Sum(list):
    '''A sum is a list of terms with integer coefficients.
Terms are arbitrary object with the only restriction that they
must be hashable.'''

    def __init__(self, elts, coeffs=None):
        if coeffs == None:
            coeffs = [1 for e in elts]
        else:
            coeffs = coeffs
        self.coeffs = []
        for i,e in enumerate(elts):
            if isinstance(e,Sum):
                for j,p in enumerate(e):
                    self.append(p)
                    self.coeffs.append( coeffs[i] * e.coeffs[j]
)
            else:
                self.append(e)
                self.coeffs.append(coeffs[i])


    def diff(self,i,n=1):
        if n == 1:
            return Sum(  [ e.diff(i) for e in self ] ,
self.coeffs ).collect()
        else:
```

```
            return self.diff(i,n-1).diff(i)

    def collect(self):
        new_elements = []
        new_coeffs = []
        for i,e in enumerate(self):
            if e in new_elements:
                k = new_elements.index(e)
                new_coeffs[k] += self.coeffs[i]
            else:
                new_elements.append(e)
                new_coeffs.append(self.coeffs[i])
        return Sum(new_elements,new_coeffs)



    def _latex_(self):
        terms = []
        for i,e in enumerate(self):
            if self.coeffs[i] == 1:
                terms.append( e._latex_() )
            else:
                terms.append( str(self.coeffs[i]) + ' ' +
e._latex_() )
        s = str.join(' + ', terms)
        return s
```

```
class Bracket(list):
    '''This class represent a multilinear function. Its content
is stored as a list of tuples.
    For instance, "[ ('a','e'),('e',) ]" would represent "f^{2}
[V_ae,V_e]".
    Derivatives with respect to any variable 'x' is computed by
added successively 'x' to each of the tuples, and to add a new
tuple containing only 'x'. Hence the result is a list of
brackets. Each bracket of the resulting sum is also reordered
in lexical order w.r.t. to its tuples.
    '''

    def diff(self,i):
        new_brackets = [ ]
        for k in range(len(self)):
            n = Bracket(self)
            n[k] = n[k] + (i,)
            n.reorder()
            new_brackets.append(n)
        b = Bracket( tuple( self + [(i,)] ) )
        b.reorder()
        new_brackets.append( b )
```

```
        return Sum(new_brackets).collect()

    def reorder(self):
        for i,e in enumerate(self):
            self[i] = tuple( sorted(e) )
        self.sort()
        return None

    def is_ascending(self):
        l = []
        [l.extend(list(a)) for a in self]
        for i in range(1,len(l)):
            if l[i] < l[i-1]:
                return False
        return True

    def order_swaps(self):
        l = []
        [l.extend(list(a)) for a in self]
        [junk,swaps] = quick(l)
        return swaps



    def __hash__(self):
        return tuple(self).__hash__()

#    def __eq__(self,x):
#        print 'Comparing brackets'
#        if self.__hash__() == x.__hash__():
#            return True
#        else:
#            return False
    def _latex_(self):
        if True:
            n = len(self)
            s = "f^{{({n})}}[{content}]"
            content = []
            for el in self:
                index = str.join(' ',  [ str(e) for e in el ]
)
                content.append(
'V_{{{index}}}'.format(index=index) )
            content = str.join(',',content)
            s = s.format(n=n,content=content)
            s = s.replace('s','\\sigma')
            return s
```

```
%hide
```

```
# examples
s = '''
Bracket([]);
Bracket([('a',)]);
Bracket([('z',)]);
Bracket([('a',),('a','e')]);
Bracket([('a',),('a','e')]).diff('a');
Bracket([('a',),('a','e')]).diff('z')

'''

exemples = [e.strip() for e in s.split(';')]
exemples = [ [e, eval(e)] for e in exemples ]
html.table([['', 'Examples']] +exemples)
```

	Examples
Bracket([])	$f^{(0)}[]$
Bracket([('a',)])	$f^{(1)}[V_a]$
Bracket([('z',)])	$f^{(1)}[V_z]$
Bracket([('a',),('a','e')])	$f^{(2)}[V_a, V_{ae}]$
Bracket([('a',),('a','e')]).diff('a')	$f^{(2)}[V_{aa}, V_{ae}] + f^{(2)}[V_a, V_{a}$
Bracket([('a',),('a','e')]).diff('z')	$f^{(2)}[V_{ae}, V_{az}] + f^{(2)}[V_a, V_{a}$

```
class BracketG(Bracket):

    def diff(self,i):
        new_brackets = [ ]
        for k in range(len(self)):
            if self[k] not in ( ('t',), ('y',), ('z',) ):
                n = BracketG(self)
                n[k] = n[k] + (i,)
                n.reorder()
                new_brackets.append(n)
        if i not in ('p','s'):
            b = BracketG( tuple( self + [(i,)] ) )
            b.reorder()
            new_brackets.append( b )
        elif i == 'p':
            b = BracketG( tuple( self + [(i,)] ) )
            b1 = BracketG( tuple( self + [('t',)] ) )
            b.reorder()
            b1.reorder()
```

```
            new_brackets.extend( [b,b1] )
        elif i == 's':
            b = BracketG( tuple( self + [(i,)] ) )
            b1 = BracketG( tuple( self + [('y',)] ) )
            b2 = BracketG( tuple( self + [('z',)] ) )
            b.reorder()
            b1.reorder()
            b2.reorder()
            new_brackets.append( b )
            new_brackets.append( b1 )
            new_brackets.append( b2 )
        return Sum(new_brackets)

    def _latex_(self,type='f'):
        if True:
            n = len(self)
            s = 'g^{{( {n} )}}_{{ {ind} }}[ {content} ]'
            ind = []
            for i in self:
                if i == ('t',):
                    ind.append('p')
                elif i == ('y',):
                    ind.append('u')
                elif i == ('z',):
                    ind.append('\\sigma')
                else:
                    ind.append('a')
            ind = str.join('',ind)
            content = []
            for i in self:
                if i == ('t',):
                    content.append('I_p')
                elif i == ('y',):
                    content.append('q')
                elif i == ('z',):
                    content.append('I_s')
                else:
                    content.append( 'g_{{ {0} }}'.format(
str.join('', i) ) )
            content = str.join(',',content)
            content = content.replace('s','\\sigma
').replace('q','\\epsilon')
            return s.format(n=n,ind=ind,content=content)
```

```
%hide
# examples
s = '''
BracketG([]);
BracketG([('a',)]);
```

```
BracketG([('z',)]);
BracketG([('a',),('a','e')]);
BracketG([('a',),('a','e')]).diff('a');
BracketG([('a',),('a','e')]).diff('s')

'''

exemples = [e.strip() for e in s.split(';')]
exemples = [ [e, eval(e)] for e in exemples ]
html.table([['', 'Examples']] +exemples)
```

	Examples
BracketG([])	$g^{(0)}[]$
BracketG([('a',)])	$g_{a}^{(1)}[g_{a}]$
BracketG([('z',)])	$g_{\sigma}^{(1)}[I_{\sigma}]$
BracketG([('a',),('a','e')])	$g_{aa}^{(2)}[g_{a}, g_{ae}]$
BracketG([('a',),('a','e')]).diff('a')	$g_{aa}^{(2)}[g_{aa}, g_{ae}] + g_{aa}^{(2)}[g_{a}, g_{a}$
BracketG([('a',),('a','e')]).diff('s')	$g_{aa}^{(2)}[g_{ae}, g_{a\sigma}] + g_{aa}^{(2)}[g_{a}, g_{a}$

```
def kill_s(sum_expr):
    '''Simplifies expressions using g_s=0.'''
    bras = []
    coeffs = []
    for i,expr in enumerate(sum_expr):
        coeff = sum_expr.coeffs[i]
        valid = True
        for e in expr:
            if e.count('s') == 1:
                valid = False
                pass
        if expr.count(('z',)) == 1:
            valid = False
        if valid:
            bras.append(expr)
            coeffs.append(coeff)
    return Sum(bras,coeffs)
```

```
s = Sum([ BracketG([]) ])
expr = s.diff('a').diff('s',1)
expr
```

$$g_{a}^{(1)}[g_{a\sigma}] + g_{aa}^{(2)}[g_{a}, g_{\sigma}] + g_{au}^{(2)}[g_{a}, \epsilon] + g_{a\sigma}^{(2)}[g_{a}, I_{\sigma}]$$

```
resp = kill_s(expr)
resp
```

$$g_{au}^{(2)}[g_{a}, \epsilon]$$

```
%time
test = s.diff('s').diff('s')
test
```

$$g_{a}^{(1)}[g_{\sigma\sigma}] + g_{aa}^{(2)}[g_{\sigma}, g_{\sigma}] + 2g_{au}^{(2)}[g_{\sigma}, \epsilon] + 2g_{a\sigma}^{(2)}[g_{\sigma}, I_{\sigma}] + g_{uu}^{(2)}[\epsilon, \epsilon] + 2g_{u\sigma}^{(2)}[\epsilon, I_{\sigma}] + g_{\sigma\sigma}^{(2)}[I_{\sigma}$$

```
CPU time: 0.00 s,  Wall time: 0.00 s
```

```
base = Sum([ Bracket([]) ])
l = base.diff('a').diff('s').diff('s').collect()
l
```

$$f^{(1)}[V_{a\sigma\sigma}] + 2f^{(2)}[V_{a\sigma}, V_{\sigma}] + f^{(2)}[V_{a}, V_{\sigma\sigma}] + f^{(3)}[V_{a}, V_{\sigma}, V_{\sigma}]$$

```
base_g = Sum([BracketG([])])
l = base_g.diff('a').diff('s').diff('s').collect()
l
```

$$g_{a}^{(1)}[g_{a\sigma\sigma}] + 2g_{aa}^{(2)}[g_{a\sigma}, g_{\sigma}] + 2g_{au}^{(2)}[g_{a\sigma}, \epsilon] + 2g_{a\sigma}^{(2)}[g_{a\sigma}, I_{\sigma}] + g_{aa}^{(2)}[g_{a}, g_{\sigma\sigma}] + g_{aaa}^{(3)}[g_{a}, g_{\sigma},$$

```
kill_s(l)
```

$$g_{a}^{(1)}[g_{a\sigma\sigma}] + g_{aa}^{(2)}[g_{a}, g_{\sigma\sigma}] + g_{auu}^{(3)}[g_{a}, \epsilon, \epsilon] + g_{a\sigma\sigma}^{(3)}[g_{a}, I_{\sigma}, I_{\sigma}]$$

```
def get_nondecreasing_indices(args,order):
    indices = [[(i,) for i in args]]
    for o in range(2,order+1):
        pindices = indices[-1]
        cindices = []
        for e in args:
            for p in pindices:
                if p[-1] <= e:
                    cindices += [ p + (e,) ]
        indices.append(cindices)
    return indices
```

```
args = ('a','e','s')
nd_inds = get_nondecreasing_indices(args,3)
tab = []
for ordind in nd_inds:
    for ind in ordind:
        b = base
        for i in ind:
            b=b.diff(i).collect()
        tab.append([ind,b])
```

```
html.table( tab )
```

(a)	$f^{(1)}[V_a]$
(e)	$f^{(1)}[V_e]$
(s)	$f^{(1)}[V_\sigma]$
(a, a)	$f^{(1)}[V_{aa}] + f^{(2)}[V_a, V_a]$
(a, e)	$f^{(1)}[V_{ae}] + f^{(2)}[V_a, V_e]$
(e, e)	$f^{(1)}[V_{ee}] + f^{(2)}[V_e, V_e]$
(a, s)	$f^{(1)}[V_{a\sigma}] + f^{(2)}[V_a, V_\sigma]$
(e, s)	$f^{(1)}[V_{e\sigma}] + f^{(2)}[V_e, V_\sigma]$
(s, s)	$f^{(1)}[V_{\sigma\sigma}] + f^{(2)}[V_\sigma, V_\sigma]$
(a, a, a)	$f^{(1)}[V_{aaa}] + 3f^{(2)}[V_a, V_{aa}] + f^{(3)}[V_a, V_a, V_a]$
(a, a, e)	$f^{(1)}[V_{aae}] + f^{(2)}[V_{aa}, V_e] + 2f^{(2)}[V_a, V_{ae}] + f^{(3)}[V_a, V_a, V_e]$
(a, e, e)	$f^{(1)}[V_{aee}] + 2f^{(2)}[V_{ae}, V_e] + f^{(2)}[V_a, V_{ee}] + f^{(3)}[V_a, V_e, V_e]$
(e, e, e)	$f^{(1)}[V_{eee}] + 3f^{(2)}[V_e, V_{ee}] + f^{(3)}[V_e, V_e, V_e]$
(a, a, s)	$f^{(1)}[V_{aa\sigma}] + f^{(2)}[V_{aa}, V_\sigma] + 2f^{(2)}[V_a, V_{a\sigma}] + f^{(3)}[V_a, V_a, V_\sigma]$
(a, e, s)	$f^{(1)}[V_{ae\sigma}] + f^{(2)}[V_{ae}, V_\sigma] + f^{(2)}[V_{a\sigma}, V_e] + f^{(2)}[V_a, V_{e\sigma}] + f^{(3)}[V_a, V_e, V$
(e, e, s)	$f^{(1)}[V_{ee\sigma}] + f^{(2)}[V_{ee}, V_\sigma] + 2f^{(2)}[V_e, V_{e\sigma}] + f^{(3)}[V_e, V_e, V_\sigma]$
(a, s, s)	$f^{(1)}[V_{a\sigma\sigma}] + 2f^{(2)}[V_{a\sigma}, V_\sigma] + f^{(2)}[V_a, V_{\sigma\sigma}] + f^{(3)}[V_a, V_\sigma, V_\sigma]$
(e, s, s)	$f^{(1)}[V_{e\sigma\sigma}] + 2f^{(2)}[V_{e\sigma}, V_\sigma] + f^{(2)}[V_e, V_{\sigma\sigma}] + f^{(3)}[V_e, V_\sigma, V_\sigma]$
(s, s, s)	$f^{(1)}[V_{\sigma\sigma\sigma}] + 3f^{(2)}[V_\sigma, V_{\sigma\sigma}] + f^{(3)}[V_\sigma, V_\sigma, V_\sigma]$

```
args = ('a','e','s')
nd_inds = get_nondecreasing_indices(args,3)
tab_g = []
for ordind in nd_inds:
    for ind in ordind:
        b = base_g
        for i in ind:
            b=b.diff(i).collect()
```

```
            #b = kill_s(b)
        tab_g.append([ind,kill_s(b)])
html.table( tab_g )
```

(a)	$g^{(1)}_{a}[g_{a}]$
(e)	$g^{(1)}_{a}[g_{e}]$
(s)	$g^{(1)}_{u}[\epsilon]$
(a, a)	$g^{(1)}_{a}[g_{aa}] + g^{(2)}_{aa}[g_{a}, g_{a}]$
(a, e)	$g^{(1)}_{a}[g_{ae}] + g^{(2)}_{aa}[g_{a}, g_{e}]$
(e, e)	$g^{(1)}_{a}[g_{ee}] + g^{(2)}_{aa}[g_{e}, g_{e}]$
(a, s)	$g^{(2)}_{au}[g_{a}, \epsilon]$
(e, s)	$g^{(2)}_{au}[g_{e}, \epsilon]$
(s, s)	$g^{(1)}_{a}[g_{\sigma\sigma}] + g^{(2)}_{uu}[\epsilon, \epsilon] + g^{(2)}_{\sigma\sigma}[I_{\sigma}, I_{\sigma}]$
(a, a, a)	$g^{(1)}_{a}[g_{aaa}] + 3g^{(2)}_{aa}[g_{a}, g_{aa}] + g^{(3)}_{aaa}[g_{a}, g_{a}, g_{a}]$
(a, a, e)	$g^{(1)}_{a}[g_{aae}] + g^{(2)}_{aa}[g_{aa}, g_{e}] + 2g^{(2)}_{aa}[g_{a}, g_{ae}] + g^{(3)}_{aaa}[g_{a}, g_{a}, g_{e}]$
(a, e, e)	$g^{(1)}_{a}[g_{aee}] + 2g^{(2)}_{aa}[g_{ae}, g_{e}] + g^{(2)}_{aa}[g_{a}, g_{ee}] + g^{(3)}_{aaa}[g_{a}, g_{e}, g_{e}]$
(e, e, e)	$g^{(1)}_{a}[g_{eee}] + 3g^{(2)}_{aa}[g_{e}, g_{ee}] + g^{(3)}_{aaa}[g_{e}, g_{e}, g_{e}]$
(a, a, s)	$g^{(2)}_{au}[g_{aa}, \epsilon] + g^{(3)}_{aau}[g_{a}, g_{a}, \epsilon]$
(a, e, s)	$g^{(2)}_{au}[g_{ae}, \epsilon] + g^{(3)}_{aau}[g_{a}, g_{e}, \epsilon]$
(e, e, s)	$g^{(2)}_{au}[g_{ee}, \epsilon] + g^{(3)}_{aau}[g_{e}, g_{e}, \epsilon]$
(a, s, s)	$g^{(1)}_{a}[g_{a\sigma\sigma}] + g^{(2)}_{aa}[g_{a}, g_{\sigma\sigma}] + g^{(3)}_{auu}[g_{a}, \epsilon, \epsilon] + g^{(3)}_{a\sigma\sigma}[g_{a}, I_{\sigma}, I_{\sigma}]$
(e, s, s)	$g^{(1)}_{a}[g_{e\sigma\sigma}] + g^{(2)}_{aa}[g_{e}, g_{\sigma\sigma}] + g^{(3)}_{auu}[g_{e}, \epsilon, \epsilon] + g^{(3)}_{a\sigma\sigma}[g_{e}, I_{\sigma}, I_{\sigma}]$
(s, s, s)	$g^{(1)}_{a}[g_{\sigma\sigma\sigma}] + 3g^{(2)}_{au}[g_{\sigma\sigma}, \epsilon] + g^{(3)}_{uuu}[\epsilon, \epsilon, \epsilon] + 3g^{(3)}_{u\sigma\sigma}[\epsilon, I_{\sigma}, I_{\sigma}] + g^{(3)}_{\sigma\sigma\sigma}[I_{\sigma}, I_{\sigma}, I_{\sigma}]$

```
args = ('a','e','s')
nd_inds = get_nondecreasing_indices(args,3)
tab = []
for ordind in nd_inds:
    for ind in ordind:
```

```
        b = base
        for i in ind:
            b=b.diff(i).collect()
        tab.append([ind,b])
```

```
args = ('a','e','s')
nd_inds = get_nondecreasing_indices(args,3)
tab_g = []
for ordind in nd_inds:
    for ind in ordind:
        b = base_g
        for i in ind:
            b=b.diff(i).collect()
            #b = kill_s(b)
        tab_g.append([ind,kill_s(b)])
html.table( tab_g )
```

(a)	$g_a^{(1)}[g_a]$
(e)	$g_a^{(1)}[g_e]$
(s)	$g_u^{(1)}[\epsilon]$
(a, a)	$g_a^{(1)}[g_{aa}] + g_{aa}^{(2)}[g_a, g_a]$
(a, e)	$g_a^{(1)}[g_{ae}] + g_{aa}^{(2)}[g_a, g_e]$
(e, e)	$g_a^{(1)}[g_{ee}] + g_{aa}^{(2)}[g_e, g_e]$
(a, s)	$g_{au}^{(2)}[g_a, \epsilon]$
(e, s)	$g_{au}^{(2)}[g_e, \epsilon]$
(s, s)	$g_a^{(1)}[g_{\sigma\sigma}] + g_{uu}^{(2)}[\epsilon, \epsilon] + g_{\sigma\sigma}^{(2)}[I_\sigma, I_\sigma]$
(a, a, a)	$g_a^{(1)}[g_{aaa}] + 3g_{aa}^{(2)}[g_a, g_{aa}] + g_{aaa}^{(3)}[g_a, g_a, g_a]$
(a, a, e)	$g_a^{(1)}[g_{aae}] + g_{aa}^{(2)}[g_{aa}, g_e] + 2g_{aa}^{(2)}[g_a, g_{ae}] + g_{aaa}^{(3)}[g_a, g_a, g_e]$
(a, e, e)	$g_a^{(1)}[g_{aee}] + 2g_{aa}^{(2)}[g_{ae}, g_e] + g_{aa}^{(2)}[g_a, g_{ee}] + g_{aaa}^{(3)}[g_a, g_e, g_e]$
(e, e, e)	$g_a^{(1)}[g_{eee}] + 3g_{aa}^{(2)}[g_e, g_{ee}] + g_{aaa}^{(3)}[g_e, g_e, g_e]$
(a, a, s)	$g_{au}^{(2)}[g_{aa}, \epsilon] + g_{aau}^{(3)}[g_a, g_a, \epsilon]$
(a, e, s)	$g_{au}^{(2)}[g_{ae}, \epsilon] + g_{aau}^{(3)}[g_a, g_e, \epsilon]$
(e, e, s)	$g_{au}^{(2)}[g_{ee}, \epsilon] + g_{aau}^{(3)}[g_e, g_e, \epsilon]$
(a, s, s)	$g_a^{(1)}[g_{a\sigma\sigma}] + g_{aa}^{(2)}[g_a, g_{\sigma\sigma}] + g_{auu}^{(3)}[g_a, \epsilon, \epsilon] + g_{a\sigma\sigma}^{(3)}[g_a, I_\sigma, I_\sigma]$
(e, s, s)	$g_a^{(1)}[g_{e\sigma\sigma}] + g_{aa}^{(2)}[g_e, g_{\sigma\sigma}] + g_{auu}^{(3)}[g_e, \epsilon, \epsilon] + g_{a\sigma\sigma}^{(3)}[g_e, I_\sigma, I_\sigma]$
(s, s, s)	$g_a^{(1)}[g_{\sigma\sigma\sigma}] + 3g_{au}^{(2)}[g_{\sigma\sigma}, \epsilon] + g_{uuu}^{(3)}[\epsilon, \epsilon, \epsilon] + 3g_{u\sigma\sigma}^{(3)}[\epsilon, I_\sigma, I_\sigma] + g_{\sigma\sigma\sigma}^{(3)}[I_\sigma, I_\sigma, I_$

```
class BracketG(Bracket):

    def diff(self,i):
        new_brackets = [ ]
        for k in range(len(self)):
            if self[k] not in ( ('t',), ('y',), ('z',) ):
                n = BracketG(self)
                n[k] = n[k] + (i,)
```

```
                n.reorder()
                new_brackets.append(n)
        if i not in ('p','s'):
            b = BracketG( tuple( self + [(i,)] ) )
            b.reorder()
            new_brackets.append( b )
        elif i == 'p':
            b = BracketG( tuple( self + [(i,)] ) )
            b1 = BracketG( tuple( self + [('t',)] ) )
            b.reorder()
            b1.reorder()
            new_brackets.extend( [b,b1] )
        elif i == 's':
            b = BracketG( tuple( self + [(i,)] ) )
            b1 = BracketG( tuple( self + [('y',)] ) )
            b2 = BracketG( tuple( self + [('z',)] ) )
            b.reorder()
            b1.reorder()
            b2.reorder()
            new_brackets.append( b )
            new_brackets.append( b1 )
            new_brackets.append( b2 )
        return Sum(new_brackets)

    def _latex_(self,type='f'):
        if True:
            n = len(self)
            s = 'g^{{( {n} )}}_{{ {ind} }}[ {content} ]'
            ind = []
            for i in self:
                if i == ('t',):
                    ind.append('p')
                elif i == ('y',):
                    ind.append('u')
                elif i == ('z',):
                    ind.append('\\sigma')
                else:
                    ind.append('a')
            ind = str.join('',ind)
            content = []
            for i in self:
                if i == ('t',):
                    content.append('I_p')
                elif i == ('y',):
                    content.append('q')
                elif i == ('z',):
                    content.append('I_s')
                else:
                    content.append( 'g_{{ {0} }}'.format(
```

```
str.join('', i) ) )
            content = str.join(',',content)
            content = content.replace('s','\\sigma
').replace('q','\\epsilon')
            return s.format(n=n,ind=ind,content=content)
```

```
s = Sum([ BracketG([]) ])
expr = s.diff('a').diff('s')
expr
```

$$g_{a}^{(1)}[g_{a\sigma}] + g_{aa}^{(2)}[g_a, g_\sigma] + g_{au}^{(2)}[g_a, \epsilon] + g_{a\sigma}^{(2)}[g_a, I_\sigma]$$

```
resp = kill_s(expr)
```

```
resp
```

$$g_{au}^{(2)}[g_a, \epsilon]$$

```
kill_s( expr )
```

$$g_{au}^{(2)}[g_a, \epsilon]$$

```
def get_K(expr):
    resp =  Sum([ h for h in expr if len(h) > 1])
    other = Sum([ h for h in expr if len(h) == 1])
    return [resp,other]
```

```
[resp,other] = get_K(l)
other
```

$$g_{a}^{(1)}[g_{a\sigma\sigma}]$$

```
def get_L(expr):
    n = sum([len(el) for el in expr[0]])
    special = BracketG( [('a',)] ) * n
    resp =  Sum([ h for h in expr if (len(h)>1) and
(h!=special) ])
    other = Sum([ h for h in expr if (len(h) == 1) or
h==special])
    return [resp,other]
```

```
t = BracketG([]).diff('a').diff('a').diff('a')
get_L(t)
```

$$\left[g_{aa}^{(2)}[g_a, g_{aa}], g_{a}^{(1)}[g_{aaa}] + g_{aaa}^{(3)}[g_a, g_a, g_a]\right]$$

```
[resp,other] = get_L(l)
resp
```

$$g_{aa}^{(2)}[g_{a\sigma}, g_\sigma] + g_{au}^{(2)}[g_{a\sigma}, \epsilon] + g_{a\sigma}^{(2)}[g_{a\sigma}, I_\sigma] + g_{aa}^{(2)}[g_a, g_{\sigma\sigma}] + g_{aaa}^{(3)}[g_a, g_\sigma, g_\sigma] + g_{aau}^{(3)}[g_a, g_a$$

```
other
```

$$g_a^{(1)}[g_{a\sigma\sigma}] + g_{aaa}^{(3)}[g_a, g_\sigma, g_\sigma] + g_{aau}^{(3)}[g_a, g_\sigma, \epsilon] + g_{aa\sigma}^{(3)}[g_a, g_\sigma, I_\sigma] + g_{auu}^{(3)}[g_a, \epsilon, \epsilon] + g_{au\sigma}^{(3)}[g_a$$

```
def compile_bracket(bra):
    if isinstance(bra,Bracket):
        content = [ "V_{0}".format( str.join("",e) ) for e in
bra]
        txt = 'mdot([{0}])'.format( str.join(",",content))
    elif isinstance(bra,BracketG):
        content = [ "g_{0}".format( str.join("",e) ) for e in
bra]
        txt = 'mdot([{0}])'.format( str.join(",",content))
    if not bra.is_ascending():
        l = []
        for t in bra:
            for e in t:
                l.append(e)
        [nl, exchanges] = quick(l)
        swp = [".swapaxes({0},{1})".format(e[0],e[1]) for e in
exchanges]
        txt += str.join('',swp)
    return txt
```

```
b = Bracket([('a','b'),('a','b')])
compile_bracket( b )
```

mdot([V_ab,V_ab]).swapaxes(1,2)

```
b = BracketG([('a','b'),('a','b')])
compile_bracket( b )
```

mdot([V_ab,V_ab]).swapaxes(1,2)

```
def compile_sum(s):
    resp = str.join('+',[ '{0}*
{1}'.format(s.coeffs[n],compile_bracket(bra)) for n,bra in
enumerate(s) ])
    return resp
```

```
compile_sum( Bracket([]).diff('a').diff('a').diff('a') )
```

1*mdot([V_aaa])+3*mdot([V_a,V_aa])+1*mdot([V_a,V_a,V_a])

```
def compute_derivatives(n,vars,):
    eqs_f = [ Sum([Bracket([])]) ]
    eqs_f.append( [ eqs_f[-1].diff(v) for v in vars ] )
    eqs_g = [ Sum([BracketG([])]) ]
    eqs_g.append( [ eqs_g[-1].diff(v) for v in vars ] )
    ndt = [[],[(v,) for v in vars]]
    print 'ok'
```

```
    for i in range(n):
        new_eqs_f = []
        new_eqs_g = []
        new_ndt = []
        o_eqs_f = eqs_f[-1]
        o_eqs_g = eqs_g[-1]
        o_ndt = ndt[-1]
        for i,e in enumerate(o_eqs_f):
            ind = o_ndt[i]
            for v in vars:
                if v>= ind[-1]:
                    new_eqs_f.append( e.diff(v).collect() )
        for i,e in enumerate(o_eqs_g):
            ind = o_ndt[i]
            for v in vars:
                if v>= ind[-1]:
                    new_eqs_g.append( e.diff(v).collect() )
                    new_ndt.append( ind + (v,) )
        eqs_f.append(new_eqs_f)
        eqs_g.append(new_eqs_g)
        ndt.append(new_ndt)
    return [eqs_f,eqs_g,ndt]
```

```
[eqs_f,eqs_g,ndt] = compute_derivatives(4,('a','e'))
```

ok

```
eqs_g
```

$$\left[g^{(0)}[], \left[g_a^{(1)}[g_a], g_a^{(1)}[g_e]\right], \left[g_a^{(1)}[g_{aa}] + g_{aa}^{(2)}[g_a, g_a], g_a^{(1)}[g_{ae}] + g_{aa}^{(2)}[g_a, g_e], g_a^{(1)}[g_{ee}] +\right.$$

```
[b for b in eqs[-1][2] if not b.is_ascending()]
```

$$[f^{(2)}[V_{aae}, V_{ae}], f^{(3)}[V_a, V_{ae}, V_{ae}]]$$

```
def write_order(order,vars,eqs_f,eqs_g,ndt):
    code = "{br}Computing order {order}\n\n".format(br='\n#' +
10*'-' ,order=order)
    code += 'order = {0}\n'.format(order)
    for k in range(len(eqs[order])):
        eq_f = eqs_f[order][k]
        eq_g = eqs_g[order][k]
        inds = ndt[order][k]
        code += '\n#--- Computing derivatives
{0}\n\n'.format(inds)
        print eq_f
        [K_eq,K_other] = get_K(eq_f)
        K_name = 'K_'+str.join('',inds)
        lhs = K_name
        code += '{lhs} =
{rhs}\n'.format(lhs=lhs,rhs=compile_sum(K_eq))
```

```
        if k == 0:
            [L_eq,L_other] = get_L(eq_g)
        else:
            L_eq = K_eq
            L_other = K_other
        L_name = 'L_'+str.join('',inds)
        lhs = L_name
        code += '{lhs} =
{rhs}\n\n'.format(lhs=lhs,rhs=compile_sum(L_eq))
        if k == 0:
            code += '#We need to solve the infamous sylvester
equation\n'
            code += '#A = f_a + sdot(f_d,g_a)\n'
            code += '#B = f_d\n'
            code += '#C = g_a\n'
            code += 'D = {K} + sdot(f_d,
{L})\n'.format(K=K_name,L=L_name)
            code += 'g_{sub} =
solve_sylvester(A,B,C,D)\n'.format(sub = str.join('',inds))
        else:
            code += '#We solve A*X + const = 0\n'
            code += 'const = sdot(f_d,{L}) +
{K}\n'.format(L=L_name,K=K_name)
            code += 'g_{sub} = - sdot(A_inv,
const)\n'.format(sub=str.join('',inds))
        if True:
            sizes = str.join(',', ['n_'+v for v in vars] )
            code += '\nif order < max_order:\n'
            code += '    Y = {L}{other}\n'.format(L = L_name,
other = compile_sum(L_other) )
            code += '    Z =
g_{sub}\n'.format(sub=str.join('',inds))
            code += '    V_{sub} = build_V(Y,Z,
({sizes}))\n'.format(sub = str.join('',inds), sizes=sizes)
#            code += 'end\n'
    return code

def gen_code(max_order,vars):
    [eqs_f,eqs_g,ndt] = compute_derivatives(max_order,vars)

    code = ''
#    code = 'max_order = {0}\n'.format(max_order)
    for o in range(2,max_order+1):
        code += write_order(o,vars,eqs_f,eqs_g,ndt)
    return code
```

```
%time
code = gen_code(3,('a','e'))
```

```
ok
[[('a', 'a')], [('a',), ('a',)]]
[[('a', 'e')], [('a',), ('e',)]]
[[('e', 'e')], [('e',), ('e',)]]
[[('a', 'a', 'a')], [('a',), ('a', 'a')], [('a',), ('a',), ('a
[[('a', 'a', 'e')], [('a', 'a'), ('e',)], [('a',), ('a', 'e')
[('a',), ('a',), ('e',)]]
[[('a', 'e', 'e')], [('a', 'e'), ('e',)], [('a',), ('e', 'e')
[('a',), ('e',), ('e',)]]
[[('e', 'e', 'e')], [('e',), ('e', 'e')], [('e',), ('e',), ('e

CPU time: 0.01 s,  Wall time: 0.01 s
```

```
get_K(Bracket([]).diff('a').diff('a').diff('a'))
```

$[f^{(2)}[V_a, V_{aa}] + f^{(3)}[V_a, V_a, V_a], f^{(1)}[V_{aaa}]]$

```
print code
```

```
#----------Computing order 2

order = 2

#--- Computing derivatives ('a', 'a')

K_aa = 1*mdot([V_a,V_a])
L_aa =

#We need to solve the infamous sylvester equation
#A = f_a + sdot(f_d,g_a)
#B = f_d
#C = g_a
D = K_aa + sdot(f_d,L_aa)
g_aa = solve_sylvester(A,B,C,D)

if order < max_order:
    Y = L_aa1*mdot([V_aa])+1*mdot([V_a,V_a])
    Z = g_aa
    V_aa = build_V(Y,Z,(n_a,n_e))

#--- Computing derivatives ('a', 'e')

K_ae = 1*mdot([V_a,V_e])
L_ae = 1*mdot([V_a,V_e])

#We solve A*X + const = 0
const = sdot(f_d,L_ae) + K_ae
g_ae = - sdot(A_inv, const)

if order < max_order:
    Y = L_ae1*mdot([V_ae])
```

```
    Z = g_ae
    V_ae = build_V(Y,Z,(n_a,n_e))

#--- Computing derivatives ('e', 'e')

K_ee = 1*mdot([V_e,V_e])
L_ee = 1*mdot([V_e,V_e])

#We solve A*X + const = 0
const = sdot(f_d,L_ee) + K_ee
g_ee = - sdot(A_inv, const)

if order < max_order:
    Y = L_ee1*mdot([V_ee])
    Z = g_ee
    V_ee = build_V(Y,Z,(n_a,n_e))

#----------Computing order 3

order = 3

#--- Computing derivatives ('a', 'a', 'a')

K_aaa = 1*mdot([V_a,V_aa])+1*mdot([V_a,V_a,V_a])
L_aaa = 1*mdot([V_a,V_aa])

#We need to solve the infamous sylvester equation
#A = f_a + sdot(f_d,g_a)
#B = f_d
#C = g_a
D = K_aaa + sdot(f_d,L_aaa)
g_aaa = solve_sylvester(A,B,C,D)

if order < max_order:
    Y = L_aaa1*mdot([V_aaa])+1*mdot([V_a,V_a,V_a])
    Z = g_aaa
    V_aaa = build_V(Y,Z,(n_a,n_e))

#--- Computing derivatives ('a', 'a', 'e')

K_aae = 1*mdot([V_aa,V_e])+1*mdot([V_a,V_ae])+1*mdot([V_a,V_a
L_aae = 1*mdot([V_aa,V_e])+1*mdot([V_a,V_ae])+1*mdot([V_a,V_a

#We solve A*X + const = 0
const = sdot(f_d,L_aae) + K_aae
g_aae = - sdot(A_inv, const)

if order < max_order:
    Y = L_aae1*mdot([V_aae])
    Z = g_aae
```

```
    V_aae = build_V(Y,Z,(n_a,n_e))

#--- Computing derivatives ('a', 'e', 'e')

K_aee = 1*mdot([V_ae,V_e])+1*mdot([V_a,V_ee])+1*mdot([V_a,V_e
L_aee = 1*mdot([V_ae,V_e])+1*mdot([V_a,V_ee])+1*mdot([V_a,V_e

#We solve A*X + const = 0
const = sdot(f_d,L_aee) + K_aee
g_aee = - sdot(A_inv, const)

if order < max_order:
    Y = L_aee1*mdot([V_aee])
    Z = g_aee
    V_aee = build_V(Y,Z,(n_a,n_e))

#--- Computing derivatives ('e', 'e', 'e')

K_eee = 1*mdot([V_e,V_ee])+1*mdot([V_e,V_e,V_e])
L_eee = 1*mdot([V_e,V_ee])+1*mdot([V_e,V_e,V_e])

#We solve A*X + const = 0
const = sdot(f_d,L_eee) + K_eee
g_eee = - sdot(A_inv, const)

if order < max_order:
    Y = L_eee1*mdot([V_eee])
    Z = g_eee
    V_eee = build_V(Y,Z,(n_a,n_e))
```