```
# General Case, step 4/4

var('A0,A1,A2,A3,A4,A5,A6,B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,
B13,C0,C1,C2,C3,C4,C5,D0,D1,D2,D3,D4,D5,E0,E1,E2,E3,E4,E5,E6,lambda_
1,lambda_2,lambda_3,lambda_4,alpha0,alpha1,alpha2,alpha3,alpha4,alph
a5,alpha6,alpha7,alpha8,alpha9,alpha10,alpha11,alpha12,alpha13,alpha
14,alpha15,alpha16,alpha17,alpha18,alpha19,alpha20,alpha21,alpha22,a
lpha23,zeta0,zeta1,zeta2,zeta3,zeta4,zeta5,zeta6,zeta7,mu1,mu2,mu3,m
u4,mu5,mu6,mu7,mu8,mu9,mu10,mu11,mu12,mu13,mu14,mu15,mu16,mu17,mu18,
mu19,mu20,mu21,mu22,mu23,mu24,mu25,mu26,mu27,mu28,mu29,mu30,mu31,zet
a8,zeta9,zeta10,zeta11,zeta12,zeta13,zeta14,zeta15,zeta16,zeta17,zet
a18,zeta19,zeta20,zeta21,zeta22,zeta23,zeta24,zeta25',domain='comple
x'),var('theta_0,beta,gamma',domain='positive')
n=theta_0
a0=A0
a0b=conjugate(A0)
b0=B0
b0b=conjugate(B0)
c0=C0
c0b=conjugate(C0)
d0=D0
d0b=conjugate(D0)
a1=A1
a1b=conjugate(A1)
b1=B1
b1b=conjugate(B1)
c1=C1
c1b=conjugate(C1)
d1=D1
d1b=conjugate(D1)
a2=A2
a2b=conjugate(A2)
b2=B2
b2b=conjugate(B2)
c2=C2
c2b=conjugate(C2)
d2=D2
d2b=conjugate(D2)
a3=A3
a3b=conjugate(A3)
b3=B3
b3b=conjugate(B3)
c3=C3
c3b=conjugate(C3)
d3=D3
d3b=conjugate(D3)
a4=A4
a4b=conjugate(A4)
b4=B4
b4b=conjugate(B4)
c4=C4
c4b=conjugate(C4)
d4=D4
d4b=conjugate(D4)
```

```
a5=A5
a5b=conjugate(A5)
b5=B5
b5b=conjugate(B5)
c5=C5
c5b=conjugate(C5)
d5=D5
d5b=conjugate(D5)
a6=A6
a6b=conjugate(A6)
b6=B6
b6b=conjugate(B6)
b7=B7
b7b=conjugate(B7)
b8=B8
b8b=conjugate(B8)
b9=B9
b9b=conjugate(B9)
b10=B10
b10b=conjugate(B10)
b11=B11
b11b=conjugate(B11)
b12=B12
b12b=conjugate(B12)
b13=B13
b13b=conjugate(B13)
alpha0b=conjugate(alpha0)
alpha1b=conjugate(alpha1)
alpha2b=conjugate(alpha2)
alpha3b=conjugate(alpha3)
alpha4b=conjugate(alpha4)
alpha5b=conjugate(alpha5)
alpha6b=conjugate(alpha6)
alpha7b=conjugate(alpha7)
alpha8b=conjugate(alpha8)
alpha9b=conjugate(alpha9)
alpha10b=conjugate(alpha10)
alpha11b=conjugate(alpha11)
alpha12b=conjugate(alpha12)
alpha13b=conjugate(alpha13)
alpha14b=conjugate(alpha14)
alpha15b=conjugate(alpha15)
alpha16b=conjugate(alpha16)
alpha17b=conjugate(alpha17)
alpha18b=conjugate(alpha18)
alpha19b=conjugate(alpha19)
alpha20b=conjugate(alpha20)
alpha21b=conjugate(alpha21)
alpha22b=conjugate(alpha22)
alpha23b=conjugate(alpha23)
e0=E0
e0b=conjugate(E0)
e1=E1
e1b=conjugate(E1)
```

```
e2=E2
e2b=conjugate(E2)
e3=E3
e3b=conjugate(E3)
e4=E4
e4b=conjugate(E4)
e5=E5
e5b=conjugate(E5)
e6=E6
e6b=conjugate(E6)
zeta0b=conjugate(zeta0)
zeta1b=conjugate(zeta1)
zeta2b=conjugate(zeta2)
zeta3b=conjugate(zeta3)
zeta4b=conjugate(zeta4)
zeta5b=conjugate(zeta5)
zeta6b=conjugate(zeta6)
zeta7b=conjugate(zeta7)
zeta8b=conjugate(zeta8)
zeta9b=conjugate(zeta9)
zeta10b=conjugate(zeta10)
zeta11b=conjugate(zeta11)
zeta12b=conjugate(zeta12)
zeta13b=conjugate(zeta13)
zeta14b=conjugate(zeta14)
zeta15b=conjugate(zeta15)
zeta16b=conjugate(zeta16)
zeta17b=conjugate(zeta17)
zeta18b=conjugate(zeta18)
zeta19b=conjugate(zeta19)
zeta20b=conjugate(zeta20)
zeta21b=conjugate(zeta21)
zeta22b=conjugate(zeta22)
zeta23b=conjugate(zeta23)
zeta24b=conjugate(zeta24)
zeta25b=conjugate(zeta25)




def delete_zeroes(v):
    m=matrix(SR,v.nrows(),v.ncols())
    n=0
    for i in range(v.nrows()): # We first remove the zeroes
coefficients
        if bool(v[i,0].is_zero()):
            n=n+1
        else:
            for j in range(v.ncols()):
                m[i-n,j]=v[i,j]
    return m.submatrix(0,0,v.nrows()-n,v.ncols())

def factor_simplify(v):
```

```
    m=matrix(SR,v.nrows(),v.ncols())
    n=0
    for i in range(v.nrows()):
        if bool(v[i,0].is_zero()):
            n=n+1
        else:
            for j in range(v.ncols()):
                m[i-n,j]=v[i,j]
    m=m.submatrix(0,0,v.nrows()-n,v.ncols())
    if bool(m.ncols()==3):
        n=0
        mlength=m.nrows()
        for i in range(mlength):
            if bool(m[i,0].is_zero()):
                n=n
            else:
                for j in range(mlength-i-1):
                    if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()):
                        m[i,0]=m[i,0]+m[i+1+j,0]
                        m[i+1+j,0]=0
                        n=n+1
                    else:
                        n=n
        if bool(n==0):
            return m
        else:
            return delete_zeroes(m)
    else:
        n=0
        mlentgh=m.nrows()
        for i in range(mlentgh):
            if bool(m[i,0].is_zero()):
                n=n
            else:
                for j in range(mlentgh-i-1):
                    if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()) & bool((m[i,3]-
m[i+1+j,3]).is_zero()):
                        m[i,0]=m[i,0]+m[i+1+j,0]
                        m[i+1+j,0]=0
                        m[i+1+j,1]=0
                        n=n+1
                    else:
                        n=n
        if bool(n==0):
            return m
        else:
            return delete_zeroes(m)

def matrix_full_simplify(v):
    m=matrix(SR,v.nrows(),v.ncols())
    n=0
    for i in range(v.nrows()):
```

```
        if bool(v[i,0].is_zero()):
            n=n+1
        else:
            for j in range(v.ncols()):
               m[i-n,j]=v[i,j]
    m=m.submatrix(0,0,v.nrows()-n,v.ncols())
    if bool(m.ncols()==3):
        n=0
        mlength=m.nrows()
        for i in range(mlength):
            if bool(m[i,0].is_zero()):
                n=n
            else:
                for j in range(mlength-i-1):
                    if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()):
                        m[i,0]=m[i,0]+m[i+1+j,0]
                        m[i+1+j,0]=0
                        n=n+1
                    else:
                        n=n
                m[i,0]=m[i,0].full_simplify()
        if bool(n==0):
            return m
        else:
            return delete_zeroes(m)
    else:
        n=0
        mlentgh=m.nrows()
        for i in range(mlentgh):
            if bool(m[i,0].is_zero()):
                n=n
            else:
                for j in range(mlentgh-i-1):
                    if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()) & bool((m[i,3]-
m[i+1+j,3]).is_zero()):
                        m[i,0]=m[i,0]+m[i+1+j,0]
                        m[i+1+j,0]=0
                        n=n+1
                    else:
                        n=n
                m[i,0]=m[i,0].full_simplify()
        if bool(n==0):
            return m
        else:
            return delete_zeroes(m)


def real_part(v):
    m=matrix(SR,2*v.nrows(),v.ncols())
    if bool(v.ncols()==3):
        for i in range(v.nrows()):
            m[2*i,0]=v[i,0]/2
```

```
            m[2*i,1]=v[i,1]
            m[2*i,2]=v[i,2]
            m[2*i+1,0]=conjugate(v[i,0])/2
            m[2*i+1,1]=v[i,2]
            m[2*i+1,2]=v[i,1]
        n=0
        mlength=m.nrows()
        for i in range(mlength):
            if bool(m[i,0].is_zero()):
                n=n
            else:
                for j in range(mlength-i-1):
                    if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()):
                        m[i,0]=m[i,0]+m[i+1+j,0]
                        m[i+1+j,0]=0
                        n=n+1
                    else:
                        n=n
        return delete_zeroes(m)
    else:
        for i in range(v.nrows()):
            m[2*i,0]=v[i,0]/2
            m[2*i,1]=v[i,1]
            m[2*i,2]=v[i,2]
            m[2*i,3]=v[i,3]
            m[2*i+1,0]=conjugate(v[i,0])/2
            m[2*i+1,1]=conjugate(v[i,1])
            m[2*i+1,2]=v[i,3]
            m[2*i+1,3]=v[i,2]
        n=0
        mlentgh=m.nrows()
        n=0
        mlentgh=m.nrows()
        for i in range(mlentgh):
            if bool(m[i,0].is_zero()):
                n=n
            else:
                for j in range(mlentgh-i-1):
                    if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()) & bool((m[i,3]-
m[i+1+j,3]).is_zero()):
                        m[i,0]=m[i,0]+m[i+1+j,0]
                        m[i+1+j,0]=0
                        n=n+1
                    else:
                        n=n
        return delete_zeroes(m)

def real_part2(v):
    n=v.nrows()
    m=matrix(SR,2*n,v.ncols())
    if bool(v.ncols()==3):
        for i in range(v.nrows()):
```

```
            m[i,0]=v[i,0]/2
            m[i,1]=v[i,1]
            m[i,2]=v[i,2]
            m[i+n,0]=conjugate(v[i,0])/2
            m[i+n,1]=v[i,2]
            m[i+n,2]=v[i,1]
        n=0
        mlength=m.nrows()
        for i in range(mlength):
            if bool(m[i,0].is_zero()):
                n=n
            else:
                for j in range(mlength-i-1):
                    if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()):
                        m[i,0]=m[i,0]+m[i+1+j,0]
                        m[i+1+j,0]=0
                        n=n+1
                    else:
                        n=n
        return delete_zeroes(m)
    else:
        for i in range(v.nrows()):
            m[i,0]=v[i,0]/2
            m[i,1]=v[i,1]
            m[i,2]=v[i,2]
            m[i,3]=v[i,3]
            m[i+n,0]=conjugate(v[i,0])/2
            m[i+n,1]=conjugate(v[i,1])
            m[i+n,2]=v[i,3]
            m[i+n,3]=v[i,2]
        n=0
        mlentgh=m.nrows()
        n=0
        mlentgh=m.nrows()
        for i in range(mlentgh):
            if bool(m[i,0].is_zero()):
                n=n
            else:
                for j in range(mlentgh-i-1):
                    if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()) & bool((m[i,3]-
m[i+1+j,3]).is_zero()):
                        m[i,0]=m[i,0]+m[i+1+j,0]
                        m[i+1+j,0]=0
                        n=n+1
                    else:
                        n=n
        return delete_zeroes(m)

def imaginary_part(v):
    n=v.nrows()
    m=matrix(SR,2*n,v.ncols())
    if bool(v.ncols()==3):
```

```
        for i in range(v.nrows()):
            m[i,0]=v[i,0]/2
            m[i,1]=v[i,1]
            m[i,2]=v[i,2]
            m[i+n,0]=-conjugate(v[i,0])/2
            m[i+n,1]=v[i,2]
            m[i+n,2]=v[i,1]
        n=0
        mlength=m.nrows()
        for i in range(mlength):
            if bool(m[i,0].is_zero()):
                n=n
            else:
                for j in range(mlength-i-1):
                    if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()):
                        m[i,0]=m[i,0]+m[i+1+j,0]
                        m[i+1+j,0]=0
                        n=n+1
                    else:
                        n=n
        return delete_zeroes(m)
    else:
        for i in range(v.nrows()):
            m[i,0]=v[i,0]/2
            m[i,1]=v[i,1]
            m[i,2]=v[i,2]
            m[i,3]=v[i,3]
            m[i+n,0]=-conjugate(v[i,0])/2
            m[i+n,1]=conjugate(v[i,1])
            m[i+n,2]=v[i,3]
            m[i+n,3]=v[i,2]
        n=0
        mlentgh=m.nrows()
        n=0
        mlentgh=m.nrows()
        for i in range(mlentgh):
            if bool(m[i,0].is_zero()):
                n=n
            else:
                for j in range(mlentgh-i-1):
                    if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()) & bool((m[i,3]-
m[i+1+j,3]).is_zero()):
                        m[i,0]=m[i,0]+m[i+1+j,0]
                        m[i+1+j,0]=0
                        n=n+1
                    else:
                        n=n
        return delete_zeroes(m)


def scal(v,w): #returns the scalar product of two vectors
    l = v.nrows()*w.nrows()
```

```
    m = matrix(SR,l,3)
    if bool(v.ncols()==3):
        for i in range(v.nrows()):
            for j in range(w.nrows()):
                m[i*w.nrows()+j,0]=v[i,0]*w[j,0]
                m[i*w.nrows()+j,1]=v[i,1]+w[j,1]
                m[i*w.nrows()+j,2]=v[i,2]+w[j,2]
        return m
    else:
        for i in range(v.nrows()):
           for j in range(w.nrows()):
                m[i*w.nrows()+j,0]=v[i,0]*v[i,1]*w[j,0]*w[j,1]
                m[i*w.nrows()+j,1]=v[i,2]+w[j,2]
                m[i*w.nrows()+j,2]=v[i,3]+w[j,3]
        return m


def matrix_sort(v): # the end will be used repetedly to obtain easy
reading code
    m=matrix(SR,v.nrows(),v.ncols())
    if bool(v.ncols()==3):
        for i in range(v.nrows()):
            m[i,0]=v[i,1]
            m[i,1]=v[i,2]
            m[i,2]=v[i,0]
        m=matrix(sorted(m))
        for i in range(v.nrows()):
            temp0=m[i,2]
            m[i,2]=m[i,1]
            m[i,1]=m[i,0]
            m[i,0]=temp0
        return m
    else:
        for i in range(v.nrows()):
            m[i,0]=v[i,2]
            m[i,1]=v[i,3]
            m[i,2]=v[i,0]
            m[i,3]=v[i,1]
        m=matrix(sorted(m))
        for i in range(v.nrows()):
            temp0=m[i,2]
            temp1=m[i,3]
            m[i,2]=m[i,0]
            m[i,3]=m[i,1]
            m[i,0]=temp0
            m[i,1]=temp1
        return m


def prod(scalar,vector): #returns the product of a scalar with a
vector
    l= scalar.nrows()*vector.nrows()
    m=matrix(SR,l,4)
    for i in range(scalar.nrows()):
```

```
        for j in range(vector.nrows()):
            m[i*vector.nrows()+j,0]=scalar[i,0]*vector[j,0]
            m[i*vector.nrows()+j,1]=vector[j,1]
            m[i*vector.nrows()+j,2]=scalar[i,1]+vector[j,2]
            m[i*vector.nrows()+j,3]=scalar[i,2]+vector[j,3]
    return m


def matrix_sort_wedge(v): # the end will be used repetedly to obtain
easy reading code
    m=matrix(SR,v.nrows(),v.ncols())
    for i in range(v.nrows()):
        m[i,0]=v[i,3]
        m[i,1]=v[i,4]
        m[i,2]=v[i,0]
        m[i,3]=v[i,1]
        m[i,4]=v[i,2]
    m=matrix(sorted(m))
    for i in range(v.nrows()):
        temp0=m[i,0]
        temp1=m[i,1]
        m[i,0]=m[i,2]
        m[i,1]=m[i,3]
        m[i,2]=m[i,4]
        m[i,3]=temp0
        m[i,4]=temp1
    return m

#WEDGE

def wedge_prod(v,w): #returns the wedge product of two vectors
    l= v.nrows()*w.nrows()
    m=matrix(SR,l,5)
    for i in range(v.nrows()):
        for j in range(w.nrows()):
            m[i*w.nrows()+j,0]=v[i,0]*w[j,0]
            m[i*w.nrows()+j,1]=v[i,1]
            m[i*w.nrows()+j,2]=w[j,1]
            m[i*w.nrows()+j,3]=v[i,2]+w[j,2]
            m[i*w.nrows()+j,4]=v[i,3]+w[j,3]
    return m

def diffzb_wedge(v):
    m=matrix(SR, v.nrows(),v.ncols())
    n=0
    for i in range(v.nrows()):
        if bool(v[i,4].is_zero()):
            n=n+1
        else:
            m[i-n,0]=v[i,4]*v[i,0]
            m[i-n,1]=v[i,1]
            m[i-n,2]=v[i,2]
            m[i-n,3]=v[i,3]
            m[i-n,4]=v[i,4]-1
```

```
    return m.submatrix(0,0,v.nrows()-n,v.ncols())

def diffzb_wedge2(v):
    m=matrix(SR, v.nrows(),v.ncols())
    n=0
    for i in range(v.nrows()):
        m[i-n,0]=v[i,4]*v[i,0]
        m[i-n,1]=v[i,1]
        m[i-n,2]=v[i,2]
        m[i-n,3]=v[i,3]
        m[i-n,4]=v[i,4]-1
    return m

def throw_wedge(v,bound): #THis algorithm permits to throw out all
errors larger or equal than bound
    m=matrix(SR,v.nrows(),v.ncols())
    n=0
    for i in range(v.nrows()):
            if bool(v[i,3]+v[i,4]<bound):
                m[i-n,0]=v[i,0]
                m[i-n,1]=v[i,1]
                m[i-n,2]=v[i,2]
                m[i-n,3]=v[i,3]
                m[i-n,4]=v[i,4]
            else:
                n=n+1
    return m.submatrix(0,0,v.nrows()-n,v.ncols())

def sum_matrix_wedge(v,w):
    d=v.nrows()+w.nrows()
    d1=v.nrows()
    m=matrix(SR,d,v.ncols())
    for i in range(v.nrows()):
        for j in range(v.ncols()):
            m[i,j]=v[i,j]
    for i in range(w.nrows()):
        for j in range(w.ncols()):
            m[i+d1,j]=w[i,j]
    return m

def real_part_wedge(v):
    n=v.nrows()
    m=matrix(SR,2*n,v.ncols())
    for i in range(v.nrows()):
       m[i,0]=v[i,0]/2
       m[i,1]=v[i,1]
       m[i,2]=v[i,2]
       m[i,3]=v[i,3]
       m[i,4]=v[i,4]
       m[i+n,0]=conjugate(v[i,0])/2
       m[i+n,1]=conjugate(v[i,1])
       m[i+n,2]=conjugate(v[i,2])
       m[i+n,3]=v[i,4]
       m[i+n,4]=v[i,3]
```

```
    n=0
    mlentgh=m.nrows()
    for i in range(mlentgh):
        if bool(m[i,0].is_zero()):
            n=n
        else:
            for j in range(mlentgh-i-1):
                if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()) & bool((m[i,3]-
m[i+1+j,3]).is_zero()) & bool((m[i,4]-m[i+1+j,4]).is_zero()):
                    m[i,0]=m[i,0]+m[i+1+j,0]
                    m[i+1+j,0]=0
                    n=n+1
                else:
                    n=n
    return delete_zeroes(m)

def rotation(alpha,ginvh0,phi,bound):
    temp1=throw_wedge(wedge_prod(phi,alpha),bound)
    temp2=throw_wedge(wedge_prod(ginvh0,diffzb(phi)),bound)
    return
matrix_sort_wedge(real_part_wedge(diffzb_wedge(sum_matrix(temp1,temp
2))))

#WEDGE

def bar(v):
    m=matrix(SR,v.nrows(),v.ncols())
    if bool(v.ncols()==3):
        for i in range(v.nrows()):
            m[i,0]=conjugate(v[i,0])
            m[i,1]=v[i,2]
            m[i,2]=v[i,1]
        return m
    else:
        for i in range(v.nrows()):
            m[i,0]=conjugate(v[i,0])
            m[i,1]=conjugate(v[i,1])
            m[i,2]=v[i,3]
            m[i,3]=v[i,2]
        return m

def intz(v):
    m=matrix(SR, v.nrows(),v.ncols())
    if bool(v.ncols()==3):
        for i in range(v.nrows()):
            m[i,0]=v[i,0]/(v[i,1]+1)
            m[i,1]=v[i,1]+1
            m[i,2]=v[i,2]
        return m
    else:
        for i in range(v.nrows()):
            m[i,0]=v[i,0]/(v[i,2]+1)
            m[i,1]=v[i,1]
```

```
            m[i,2]=v[i,2]+1
            m[i,3]=v[i,3]
        return m

def intzb(v):
    m=matrix(SR, v.nrows(),v.ncols())
    if bool(v.ncols()==3):
        for i in range(v.nrows()):
            m[i,0]=v[i,0]/(v[i,2]+1)
            m[i,1]=v[i,1]
            m[i,2]=v[i,2]+1
        return m
    else:
        for i in range(v.nrows()):
            m[i,0]=v[i,0]/(v[i,3]+1)
            m[i,1]=v[i,1]
            m[i,2]=v[i,2]
            m[i,3]=v[i,3]+1
        return m

def diffz(v):
    m=matrix(SR, v.nrows(),v.ncols())
    n=0
    if bool(v.ncols()==3):
        for i in range(v.nrows()):
            if bool(v[i,1].is_zero()):
                n=n+1
            else:
                m[i-n,0]=v[i,1]*v[i,0]
                m[i-n,1]=v[i,1]-1
                m[i-n,2]=v[i,2]
    else:
        for i in range(v.nrows()):
            if bool(v[i,2].is_zero()):
                n=n+1
            else:
                m[i-n,0]=v[i,2]*v[i,0]
                m[i-n,1]=v[i,1]
                m[i-n,2]=v[i,2]-1
                m[i-n,3]=v[i,3]
    return m.submatrix(0,0,v.nrows()-n,v.ncols())


def diffzb(v):
    m=matrix(SR, v.nrows(),v.ncols())
    n=0
    if bool(v.ncols()==3):
        for i in range(v.nrows()):
            if bool(v[i,2].is_zero()):
                n=n+1
            else:
                m[i-n,0]=v[i,2]*v[i,0]
                m[i-n,1]=v[i,1]
                m[i-n,2]=v[i,2]-1
```

```
    else:
        for i in range(v.nrows()):
            if bool(v[i,3].is_zero()):
                n=n+1
            else:
                m[i-n,0]=v[i,3]*v[i,0]
                m[i-n,1]=v[i,1]
                m[i-n,2]=v[i,2]
                m[i-n,3]=v[i,3]-1
    return m.submatrix(0,0,v.nrows()-n,v.ncols())


def throw(v,bound): #THis algorithm permits to throw out all errors
larger or equal than bound
    m=matrix(SR,v.nrows(),v.ncols())
    n=0
    if bool(v.ncols()==3):
        for i in range(v.nrows()):
            if bool(v[i,1]+v[i,2]<bound):
                m[i-n,0]=v[i,0]
                m[i-n,1]=v[i,1]
                m[i-n,2]=v[i,2]
            else:
                n=n+1
    else:
        for i in range(v.nrows()):
            if bool(v[i,2]+v[i,3]<bound):
                m[i-n,0]=v[i,0]
                m[i-n,1]=v[i,1]
                m[i-n,2]=v[i,2]
                m[i-n,3]=v[i,3]
            else:
                n=n+1
    return m.submatrix(0,0,v.nrows()-n,v.ncols())




def mult_scalar(l,v):
    m=matrix(SR,v.nrows(),v.ncols())
    for i in range(v.nrows()):
        m[i,0]=l*v[i,0]
        for j in range(v.ncols()-1):
            m[i,j+1]=v[i,j+1]
    return m

def sum_matrix(v,w):
    d=v.nrows()+w.nrows()
    d1=v.nrows()
    m=matrix(SR,d,v.ncols())
    for i in range(v.nrows()):
        for j in range(v.ncols()):
            m[i,j]=v[i,j]
    for i in range(w.nrows()):
```

```
        for j in range(w.ncols()):
            m[i+d1,j]=w[i,j]
    return factor_simplify(m)

def sing_quartic(ginv,h0,bound):
    mz=diffz(h0)
    mzzb=diffzb(mz)
    mzb=diffzb(h0)
    h1=scal(mzzb,h0)
    h2=scal(mz,mzb)
    return
matrix_full_simplify(throw(scal(ginv,sum_matrix(h1,mult_scalar(-1,h2
))),bound))

def sing_quartic_zero(h0,theta_0,bound):
    mz=diffz(h0)
    mzzb=diffzb(mz)
    mzb=diffzb(h0)
    h1=scal(mzzb,h0)
    h2=scal(mz,mzb)
    return
matrix_full_simplify(throw(power_divide(sum_matrix(h1,mult_scalar(-1
,h2)),theta_0-1,theta_0-1),bound))

def intQ(dphi,ginv,H,h0,bound):
    m1=throw(mult_scalar(-1,prod(scal(H,H),dphi)),bound-1)

m2=throw(mult_scalar(-2,prod(scal(H,h0),prod(ginv,bar(dphi)))),bound
-1)
    Q=intz(sum_matrix(m1,m2))
    return factor_simplify(Q)

def scal_hold(v,w): #returns the scalar product of two vectors
    l = v.nrows()*w.nrows()
    m = matrix(SR,l,3)
    if bool(v.ncols()==3):
        for i in range(v.nrows()):
            for j in range(w.nrows()):
                m[i*w.nrows()+j,0]=v[i,0].mul(w[j,0],hold=True)
                m[i*w.nrows()+j,1]=v[i,1]+w[j,1]
                m[i*w.nrows()+j,2]=v[i,2]+w[j,2]
        return m
    else:
        for i in range(v.nrows()):
           for j in range(w.nrows()):
                m[i*w.nrows()
+j,0]=v[i,0].mul(w[j,0],hold=True).mul(v[i,1].mul(w[j,1],hold=True),
hold=True)
                m[i*w.nrows()+j,1]=v[i,2]+w[j,2]
                m[i*w.nrows()+j,2]=v[i,3]+w[j,3]
        return m


def prod_hold(scalar,vector): #returns the product of a scalar with
```

```
a vector
    l= scalar.nrows()*vector.nrows()
    m=matrix(SR,l,4)
    for i in range(scalar.nrows()):
        for j in range(vector.nrows()):
            m[i*vector.nrows()
+j,0]=scalar[i,0].mul(vector[j,0],hold=True)
            m[i*vector.nrows()+j,1]=vector[j,1]
            m[i*vector.nrows()+j,2]=scalar[i,1]+vector[j,2]
            m[i*vector.nrows()+j,3]=scalar[i,2]+vector[j,3]
    return m

def factor_simplify_hold(v):
    m=matrix(SR,v.nrows(),v.ncols())
    n=0
    for i in range(v.nrows()):
        if bool(v[i,0].is_zero()):
            n=n+1
        else:
            for j in range(v.ncols()):
                m[i-n,j]=v[i,j]
    m=m.submatrix(0,0,v.nrows()-n,v.ncols())
    if bool(m.ncols()==3):
        n=0
        mlength=m.nrows()
        for i in range(mlength):
            if bool(m[i,0].is_zero()):
                n=n
            else:
                for j in range(mlength-i-1):
                    if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()):
                        m[i,0]=m[i,0].add(m[i+1+j,0],hold=True)
                        m[i+1+j,0]=0
                        n=n+1
                    else:
                        n=n
        if bool(n==0):
            return m
        else:
            return delete_zeroes(m)
    else:
        n=0
        mlentgh=m.nrows()
        for i in range(mlentgh):
            if bool(m[i,0].is_zero()):
                n=n
            else:
                for j in range(mlentgh-i-1):
                    if bool((m[i,1]-m[i+1+j,1]).is_zero()) &
bool((m[i,2]-m[i+1+j,2]).is_zero()) & bool((m[i,3]-
m[i+1+j,3]).is_zero()):
                        m[i,0]=m[i,0].add(m[i+1+j,0],hold=True)
                        m[i+1+j,0]=0
```

```
                        m[i+1+j,1]=0
                        n=n+1
                    else:
                        n=n
        if bool(n==0):
            return m
        else:
            return delete_zeroes(m)

def sum_matrix_hold(v,w):
    d=v.nrows()+w.nrows()
    d1=v.nrows()
    m=matrix(SR,d,v.ncols())
    for i in range(v.nrows()):
        for j in range(v.ncols()):
            m[i,j]=v[i,j]
    for i in range(w.nrows()):
        for j in range(w.ncols()):
            m[i+d1,j]=w[i,j]
    return factor_simplify_hold(m)



def inversion(H,h0,ginv,dzphi,bound,n):

alpha=throw(sum_matrix(diffz(H),sum_matrix(mult_scalar(2,prod(scal(H
,h0),prod(ginv,bar(dzphi)))),prod(scal(H,H),dzphi))),bound-2*n)
    phi=intz(dzphi)
    temp1=throw(prod_hold(scal(phi,phi),alpha),bound)

temp2=throw(prod_hold(mult_scalar(-2,scal(phi,alpha),phi)),bound)

temp3=throw(prod_hold(mult_scalar(-1,scal(ginv,diffzb(scal(phi,phi))
)),h0),bound)

temp4=throw(prod_hold(mult_scalar(2,scal(h0,phi)),prod(ginv,bar(dzph
i))),bound)
    final1=sum_matrix_hold(temp1,temp2)
    final2=sum_matrix_hold(temp3,temp4)
    return
factor_simplify_hold(diffzb(sum_matrix_hold(final1,final2)))


def matrix_unhold(v):
    m=matrix(SR,v.nrows(),v.ncols())
    for i in range(v.nrows()):
        m[i,0]=v[i,0].unhold()
        for j in range(v.ncols()-1):
            m[i,j+1]=v[i,j+1]
    return m

def power_divide(v,nz,nzb):
    m=matrix(SR,v.nrows(),v.ncols())
    for i in range(v.nrows()):
```

```
        for j in range(v.ncols()):
            if bool(j==v.ncols()-2):
                m[i,j]=v[i,j]-nz
            else:
                if bool(j==v.ncols()-1):
                    m[i,j]=v[i,j]-nzb
                else:
                    m[i,j]=v[i,j]
    return m

def simplify_matrix(v):
    m=matrix(SR,v.nrows(),v.ncols())
    for i in range(v.nrows()):
        for j in range(v.ncols()):
            if j==0:
                m[i,j]=v[i,j].full_simplify()
            else:
                m[i,j]=v[i,j]
    return m

def ellipse(v,n):
    m=matrix(SR,v.nrows(),v.ncols())
    for i in range(v.nrows()):
        for j in range(v.ncols()):
            if bool(j==v.ncols()-1) & bool(v[i,j]==n-1):
                m[i,0]=0
            else:
                m[i,j]=v[i,j]
    return delete_zeroes(m)

def taylor_inverse(v,order,bound):            # gives the Taylor
development up to order inverse of                  a function g
given as g(x)=1+f(x)
    m=matrix(SR,v.nrows()-1,v.ncols())
    for i in range(v.nrows()-1):
        for j in range(v.ncols()):
            if bool(j==0):
                m[i,j]=-v[i+1,j]
            else:
                m[i,j]=v[i+1,j]
    vbis=m
    temp=m
    for i in range(order-1):
        temp=scal(temp,vbis)
        m=sum_matrix(m,mult_scalar((-1)^i,throw(temp,bound)))
    add1=matrix(SR,1,v.ncols())
    add1[0,0]=1
    return factor_simplify(sum_matrix(add1,m))


def search_coefficient(h0,a,b):
    m=matrix(SR,1,1)
    for i in range(h0.nrows()):
        for j in range(h0.nrows()-i-1):
```

```
            if bool((h0[i,2]+h0[i+j+1,2]-a).is_zero()) &
bool((h0[i,3]+h0[i+j+1,3]-b).is_zero()):

m[0,0]=m[0,0]+h0[i,0]*h0[i,1]*h0[i+j+1,0]*h0[i+j+1,1]*(h0[i,2]-
h0[i+j+1,2])*(h0[i,3]-h0[i+j+1,3])
            else:
                m=m
    return m[0,0].full_simplify()

def search_coefficient_wedge(vector,a,b):
    m=matrix(SR,vector.nrows(),vector.ncols())
    n=0
    for i in range(vector.nrows()):
        if bool((vector[i,vector.ncols()-2]-a).is_zero()) &
bool((vector[i,vector.ncols()-1]-b).is_zero()):
            for j in range(vector.ncols()):
                m[i-n,j]=vector[i,j]
        else:
            n=n+1
    return m.submatrix(0,0,vector.nrows()-n,vector.ncols())

def get_coefficient_wedge(vector,a,b):

m=sum_matrix_wedge(search_coefficient_wedge(vector,a,b+1),search_coe
fficient_wedge(vector,b,a+1))
    return matrix_sort_wedge(real_part_wedge(diffzb_wedge(m)))

def search_powers(v,theta_0,bound): # with this algorithm, we obtain
all possible powers arising in the Taylor expansion of the quartic
form but delete all holomorphic coefficients
    l=v.nrows()
    c=v.ncols()
    m=matrix(SR,l*(l-1)/2,2)
    n=0
    for i in range(v.nrows()):
        for j in range(v.nrows()-i-1):
            if bool((v[i,c-2]-v[i+j+1,c-2]).is_zero()) or
bool((v[i,c-1]-v[i+j+1,c-1]).is_zero()) or
bool((v[i,c-2]+v[i+j+1,c-2]+v[i,c-1]+v[i+j+1,c-1]-2*theta_0)>bound-1
): #We check if the product is trivial or not
                n=n
            else:
                m[n,0]=v[i,c-2]+v[i+j+1,c-2]-theta_0
                m[n,1]=v[i,c-1]+v[i+j+1,c-1]-theta_0
                n=n+1
    for i in range(m.nrows()):         # We delete multiple
occurrences
            if bool(m[i,1].is_zero()): # avoids making useless
tests, as the holomorphic coefficients will be deleted by
delete_hol()
                n=n
            else:
                for j in range(m.nrows()-i-1):
                    if bool((m[i,0]-m[i+j+1,0]).is_zero()) &
```

```
bool((m[i,1]-m[i+j+1,1]).is_zero()):
                        m[i+j+1,1]=0
                    else:
                        n=n
    return delete_hol(m)

def sing_quartic2(h0,powers,n):
    m=matrix(SR,powers.nrows(),3)
    for i in range(powers.nrows()):
        m[i,0]=search_coefficient(h0,powers[i,0]+n,powers[i,1]+n) #
Recall that search_coefficient(h0,a,b) gives the coefficient
corresponding to the order z^a\z^b in Q(\h_0), not g^{-1}\otimes
Q(h0),here we actually compute g_{0}^{-1}\otimes Q(\h_0), as the
additional terms only comes from the first three holomorphic terms,
which we can easily compute by hand
        m[i,1]=powers[i,0]
        m[i,2]=powers[i,1]
    return m

def Gauss_curvature(e2u,e2uinv,bound):
    temp1=throw(scal(e2uinv,diffz(diffzb(e2u))),bound)
    temp2=throw(scal(e2uinv,scal(diffz(e2u),diffzb(e2u))),bound)
    return
factor_simplify(mult_scalar(2,sum_matrix(temp1,mult_scalar(-1,temp2)
)))

def Weingarten(dzphi,g,ginv,bound1,bound2):
    temp1=diffz(dzphi)
    dzlambda=throw(scal(ginv,diffz(g)),bound1)
    temp2=throw(mult_scalar(-1,prod(dzlambda,dzphi)),bound2)
    return factor_simplify(mult_scalar(2,sum_matrix(temp1,temp2)))

def Weingarten_simpler(dzphi,g,ginv,bound1,bound2):
    temp1=diffz(dzphi)
    dzlambda=throw(scal(ginv,diffz(g)),bound1)
    temp2=throw(mult_scalar(-1,prod(dzlambda,dzphi)),bound2)
    return
matrix_full_simplify(mult_scalar(2,sum_matrix(temp1,temp2)))


def Gauss_Weingarten(e2u,e2uinv,ginv,h0,bound):
    temp1=throw(scal(e2uinv,diffz(diffzb(e2u))),bound)
    temp2=throw(scal(e2uinv,scal(diffz(e2u),diffzb(e2u))),bound)
    K=mult_scalar(2,sum_matrix(temp1,mult_scalar(-1,temp2)))
    temp3=throw(scal(ginv,scal(h0,h0)),bound)
    return factor_simplify(throw(scal(K,temp3),bound))

#upper order next step


g2=sum_matrix(matrix([[1,n-1,n-1],[2*abs(a1)^2,n,n],
[beta,n+1,n+1]]),real_part2(matrix([[2*alpha1,n+1,n-1],
[2*alpha2,1,2*n],[2*alpha3,n+2,n-1],[2*alpha4,n+3,n-1],
[2*alpha5,n+1,n],[2*alpha6,n+2,n],[2*alpha7,3,2*n-1],
```

```
[2*alpha8,2,2*n],[2*alpha9,1,2*n+1]])))

H6=real_part2(matrix([[1,c1,2-n,0],[1,c2,3-n,0],[1,c3,4-n,0],
[1,c4,5-n,0],[1,b1,2-n,1],[1,b2,2-n,2],[1,b3,3-n,1],[1,b4,2-n,3],
[1,b5,3-n,2],[1,b6,4-n,1],[1,e1,-2*n+4,n],[1,e2,-2*n+4,n+1],
[1,e3,5-2*n,n]]))

dzphi3=sum_matrix(matrix([[1,a0,n-1,0],[1,a1,n,0],[1,a2,n+1,0],
[1,a3,n+2,0],[1,a4,n+3,0],
[1,a5,n+4,0]]),mult_scalar(1/2,matrix_sort(matrix_full_simplify(intz
b(throw(prod(g2,H6),n+4))))))

conf=matrix_sort(matrix_full_simplify(throw(scal(dzphi3,dzphi3),2*n+
4)))

g3temp=mult_scalar(2,matrix_sort(matrix_full_simplify(throw(scal(dzp
hi3,bar(dzphi3)),2*n+4))))

##latex(dzphi3),latex(conf),latex(g3temp)

phi2=sum_matrix(matrix([[1/n,a0b,0,n],[1/(n+1),a1b,0,n+1],[1/
(n+2),a2b,0,n+2],[1/(n+3),a3b,0,n+3],[1/(n+4),a4b,0,n+4],[1/
(n+5),a5b,0,n+5]]),intz(dzphi3))

##latex(phi2)

g3=sum_matrix(matrix([[1,n-1,n-1],[2*abs(a1)^2,n,n],
[beta,n+1,n+1]]),real_part2(matrix([[2*alpha1,n+1,n-1],
[2*alpha2,1,2*n],[2*alpha3,n+2,n-1],[2*alpha4,n+3,n-1],
[2*alpha5,n+1,n],[2*alpha6,n+2,n],[2*alpha7,3,2*n-1],
[2*alpha8,2,2*n],[2*alpha9,1,2*n+1],[2*alpha10,1,2*n+2],
[2*alpha11,2,2*n+1],[2*alpha12,3,2*n],[2*alpha13,4,2*n-1],
[2*alpha14,n-1,n+4],[2*alpha15,n,n+3],[2*alpha16,n+1,n+2]])))

##latex(g3)

g3inv=power_divide(taylor_inverse(power_divide(g3,n-1,n-1),3,6),n-1,
n-1)

##latex(g3inv)

#g3invtest=power_divide(taylor_inverse(power_divide(g3,n-1,n-1),4,6)
,n-1,n-1)

#zero=matrix_full_simplify(sum_matrix(g3inv,mult_scalar(-1,g3invtest
)))

#latex(zero)

h03=Weingarten_simpler(dzphi3,g3,g3inv,5,n+4)

latex(h03)

#### Main term in the quartic form for the coefficient in
```

```
z^{\theta_0}\z^{2-\theta_0}dz^4

latex(search_coefficient(h03,n+n,-n+2+n))

### vec{\alpha} 2

H6m1=throw(H6,4-n)

m1=prod(scal(H6m1,H6m1),throw(dzphi3,n+1))

h03m2=throw(h03,n+2)

H6m2=throw(H6,5-n)

g3invm2=throw(g3inv,5-2*n)

dzphi3m2=throw(dzphi3,n+2)

m2=throw(mult_scalar(2,prod(scal(H6m2,h03m2),prod(g3invm2,bar(dzphi3
m2)))),5-n)

alpha=factor_simplify(sum_matrix(diffz(H6),sum_matrix(m1,m2)))

latex(alpha)









#AFTER APHA

#phi square

phi=throw(phi2,n+5) ### is different from phi2

h0=throw(h03,n+4)  ###=h03

phi_square_temp=matrix_sort(matrix_full_simplify(throw(scal(phi,phi)
,2*n+5)))

latex(phi_square_temp)

phi_square=matrix([[-zeta0b/(n^4+4*n^3+5*n^2+2*n),0,2*n+2],
```

```
[-4*zeta1b/(n^4+6*n^3+11*n^2+6*n),0,2*n+3],[1/n^2,n,n],[alpha1b/
(n*(n+2)),n,n+2],[(-zeta2b+2*n*(n+1)*alpha3b)/
(2*(n^4+4*n^3+3*n^2)),n,n+3],[2*abs(a1)^2/(n+1)^2,n+1,n+1],[alpha5b/
((n+1)*(n+2)),n+1,n+2],[alpha1/(n*(n+2)),n+2,n],[alpha5/
((n+1)*(n+2)),n+2,n+1],[(-zeta2+2*n*(n+1)*alpha3)/
(2*(n^4+4*n^3+3*n^2)),n+3,n],[-zeta0/(n^4+4*n^3+5*n^2+2*n),2*n+2,0],
[-4*zeta1/(n^4+6*n^3+11*n^2+6*n),2*n+3,0]])

latex(phi_square)

#ginvh0

ginvh02=matrix_sort(matrix_full_simplify(throw(prod(throw(g3inv,7-2*
n),h0),6-n)))

latex(ginvh02)

ginvh0_phi_temp=matrix_sort(matrix_full_simplify(throw(scal(ginvh02,
phi),6)))

latex(ginvh0_phi_temp)

phi_square_part2=matrix([[zeta9b,0,2*n+4],[alpha7/(2*n^2),4,2*n],
[zeta8b,n,n+4],[2*zeta11b/((n+1)*(n+3)),n+1,n+3],[zeta10,n+2,n+2],
[2*zeta11/((n+1)*(n+3)),n+3,n+1],[zeta8,n+4,n],[zeta9,2*n+4,0],
[alpha7b/(2*n^2),2*n,4]])

latex(phi_square_part2)

phi_square2=sum_matrix(phi_square,phi_square_part2)

latex(phi_square2)




#alpha3

#latex(matrix_sort(throw(alpha,5-n)))   ### =vec_alpha2 as coded
below

#latex(matrix_full_simplify(sum_matrix(alpha,mult_scalar(-1,throw(al
pha,5-n)))))

vec_alpha=matrix([[2*n*(n+1)*alpha2b,a0b,0,-n+2],[zeta2b/
(n-3),a1,0,-n+3],[2*n*(n+1)*alpha2b,a1b,0,-n+3],[zeta3b,a0b,0,-n+3],
[(n-2)/(4*n)*abs(c1)^2,a0,1,-n+2],[2*zeta4b,a0b,1,-n+2],[-(n-2)/
2,c1,-n+1,0],[-(n-2)/2,b1,-n+1,1],[-(n-2)/2,b2,-n+1,2],[-
(n-2)*abs(c1)^2/(2*n),a0b,-n+1,2],[-(n-3)/2,c2,-n+2,0],
[2*zeta2,a0b,-n+2,0],[-(n-3)/2,b3,-n+2,1],[2*zeta2,a1b,-n+2,1],[-
(n-4)/2,c3,-n+3,0],[2*n*(n-4)*alpha7,a0,-n+3,0],[2*zeta5,a0b,-
n+3,0]])
```

```
#latex(vec_alpha)

vec_alpha_part2=matrix([[1,b7,0,-n+4],[1,b8,1,-n+3],[1,b9,2,-n+2],
[(n-2)*alpha2,c1,-2*n+3,n+1],[-2*(n-2)*(n-4)/
(n+1)*alpha7,a1b,-2*n+3,n+1],[1,b10,-n+1,3],[1,b11,-n+2,2],[1,b12,-
n+3,1],[1,b13,-n+4,0],[(n+1)/2*alpha2b,c1b,n,-2*n+4],[-
(n-4)*alpha7b,a1,n,-2*n+4]])

#latex(vec_alpha_part2)

vec_alpha2=sum_matrix(vec_alpha,vec_alpha_part2)

latex(vec_alpha2)




############### check the computation of \s{\vec{\alpha}}{\phi}

#alpha_phi_temp=matrix_sort(matrix_full_simplify(throw(scal(vec_alph
a2,throw(phi,n+4)),5)))

#latex(alpha_phi_temp)


alpha_phi=matrix([[(2*n-1)/(2*n*(n+1))*zeta2,2,0],[zeta7,3,0],
[(n-2)*alpha2,-n+1,n+1],[(n-2)*zeta4/(n*(n+2)),-n+1,n+2],
[(n-3)*zeta3/(2*n*(n+1)),-n+2,n+1],[(n+1)*alpha2b,n,-n+2],[zeta3b/
(2*n),n,-n+3],[zeta4b/n,n+1,-n+2]])

#latex(alpha_phi)

#test_scaling=matrix_full_simplify(real_part(diffzb(alpha_phi)))#

#latex(test_scaling)

alpha_phi_part2=matrix([[zeta13,0,4],[zeta14,1,3],[zeta15,2,2],
[zeta16,3,1],[zeta17,4,0],[zeta18,-n+1,n+3],[zeta19,-n+2,n+2],
[zeta20,-n+3,n+1],[zeta21,-n+4,n],[zeta22,n,-n+4],[zeta23,n+1,-n+3],
[zeta24,n+2,-n+2]])

#latex(alpha_phi_part2)

alpha_phi2=sum_matrix(alpha_phi,alpha_phi_part2)

latex(alpha_phi2)
```

```
#ginvh0_phi_temp=matrix_sort(matrix_full_simplify(throw(scal(ginvh02
,phi),5)))

#latex(ginvh0_phi_temp)


ginvh0_phi=matrix([[-2*abs(a1)^2/(n*(n+1)),0,2],[mu1,0,3],[mu2,0,4],
[mu3,1,2],[mu4,1,3],[mu5,2,1],[mu6,2,2],[mu7,3,1],[(n-2)/n*alpha2,-
n+1,n+2],[mu8,-n+1,n+3],[mu9,-n+2,n+2],[(n-2)*(n-4)/n^2*alpha7,-
n+3,n+1],[-(n+1)/n*alpha2b,n,-n+3],[mu10,n,-n+4],[-2*zeta0/
(n*(n+1)),n+1,-n+1],[mu11,n+1,-n+3],[mu12,n+2,-n+1],[mu13,n+2,-n+2],
[mu14,n+3,-n+1]])

latex(ginvh0_phi)

ginvh0_phi_temp2=matrix_sort(matrix_full_simplify(throw(scal(ginvh02
,phi),6)))

latex(ginvh0_phi_temp2)

ginvh0_phi_part2=matrix([[mu15,0,5],[mu16,1,4],[mu17,2,3],
[mu18,3,2],[mu19,4,1],[mu20,-n+1,n+4],[mu21,-n+2,n+3],[mu22,-
n+3,n+2],[mu23,-n+4,n+1],[mu24,n,-n+5],[mu25,n-1,-n+6],[mu26,n+1,-
n+4],[mu27,n+2,-n+3],[mu28,n+3,-n+2],[mu29,n+4,-n+1],
[mu30,2*n+2,-2*n+3],[mu31,2*n,-2*n+5]])

latex(ginvh0_phi_part2)

ginvh0_phi2=sum_matrix(ginvh0_phi,ginvh0_phi_part2)

latex(ginvh0_phi2)




end_alpha1=matrix_full_simplify(throw(prod(throw(phi_square2,2*n+4),
vec_alpha2),n+5))

end_alpha2=matrix_full_simplify(mult_scalar(-2,throw(prod(alpha_phi2
,throw(phi,n+4)),n+5)))

fate1=matrix_full_simplify(sum_matrix(end_alpha1,end_alpha2))


end_h01=matrix_full_simplify(mult_scalar(-1,throw(prod(diffzb(phi_sq
```

```
uare2),ginvh02),n+5)))

end_h02=matrix_full_simplify(mult_scalar(2,throw(prod(ginvh0_phi2,th
row(bar(dzphi3),n+4)),n+5)))  #we need each tensor up to order 5, so
the terms in |z|^{\theta_0-1+5}=|z|^{\theta_0+4} are not necessary

fate2=matrix_full_simplify(sum_matrix(end_h01,end_h02))


fate=matrix_full_simplify(sum_matrix(fate1,fate2))


end_end_end=matrix_sort(matrix_full_simplify(real_part(diffzb(fate))
))

latex(end_end_end)
```