

```
# General Case, step 1/4
```

```
var('A0,A1,A2,A3,A4,A5,A6,B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,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,alpha5,alpha6,alpha7,alpha8,alpha9,alpha10,alpha11,alpha12,alpha13,alpha14,alpha15,alpha16,alpha17,alpha18,alpha19,alpha20,alpha21,alpha22,alpha23,E_00,E_10,E_01,E_02,E_03,E_04,E_05,E_15,E_06,E_07,E_08,E_18,E_09',domain='complex'),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)
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)
e00=E_00
e10=E_10
e01=E_01
e02=E_02
e03=E_03
e04=E_04
e05=E_05
e15=E_15
e06=E_06
e07=E_07
e08=E_08
e18=E_18
e09=E_09
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)

```

```

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 matrix_sort5(v): # the end will be used repeatedly 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,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

```

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

def diffzb5(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 throw5(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 rotation(dzphi,ginv,H,h0,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-n)
    phi=intz(dzphi)
    temp1=throw5(wedge_prod(phi,alpha),bound)
    temp2=throw5(wedge_prod(prod(ginv,h0),bar(dzphi)),bound)
    return matrix_sort5(diffzb5(sum_matrix5(final1,final2)))

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+1

```

```

        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(dzphi
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_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  $z^a \setminus z^b$  in  $Q(\setminus h_0)$ , not  $g^{-1} \setminus \otimes$ 
 $Q(h_0)$ , here we actually compute  $g_{\{0\}}^{-1} \setminus \otimes Q(\setminus h_0)$ , 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))

```

```
dzphi = matrix([1,a0,n-1,0])
g = matrix([1,n-1,n-1])
ginv = matrix([1,1-n,1-n])
H = matrix([[1/2,c1,2-n,0],[1/2,c1b,0,2-n]])

latex(dzphi), latex(g), latex(H)
```