```
#Case \theta_0=4

var('A0,A1,A2,A3,A4,A5,A6,B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,C0,C1,C2
,C4,C5,D0,D1,D2,D3,D4,D5,E0,E1,E2,E3,E4,E5,E6,lambda_1,lambda_2,lamb
da_3,lambda_4,alpha0,alpha1,alpha2,alpha3,alpha4,alpha5,alpha6,alpha
7,alpha8,alpha9,alpha10,alpha11,alpha12,alpha13,alpha14,alpha15,alph
a16,alpha17,alpha18,alpha19,alpha20,alpha21,alpha22,alpha23,E_00,E_1
0,E_01,E_02,E_03,E_04,E_05,E_15,E_06,E_07,E_08,E_18,E_09,gamma2,gamm
a3,zeta0,zeta1,zeta2,zeta3,zeta4',domain='complex'),var('theta_0,bet
a,delta,C3,C6,gamma0,gamma1,gamma4',domain='positive')   ### C3 and
C6 must be real for \theta_0=4
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)
zeta0b=conjugate(zeta0)
zeta1b=conjugate(zeta1)
zeta2b=conjugate(zeta2)
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)

c6=C6 ### is real

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 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()==4):
        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()) & 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
        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()) & 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
                        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()==4):
        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()) & 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)
    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()) & 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
                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 v.ncols()==4:
        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]=v[i,2]
            m[2*i+1,2]=v[i,1]
            m[2*i+1,3]=v[i,3]
        n=0
        mlentgh=m.nrows()
        for i in range(mlentgh):
            if 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]) &
bool(m[i,2]==m[i+1+j,2]) & bool(m[i,3]==m[i+1+j,3]):
                        m[i,0]=m[i,0]+m[i+1+j,0]
                        m[i+1+j,0]=0                #We delete the
```

```
line we have just added
                        n=n+1
                    else:
                        n=n
    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,4]=v[i,4]
            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]
            m[2*i+1,4]=v[i,4]
        n=0
        mlength=m.nrows()
        for i in range(mlength):
            if 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]) &
bool(m[i,2]==m[i+1+j,2]) & bool(m[i,3]==m[i+1+j,3]) &
bool(m[i,4]==m[i+1+j,4]):
                       m[i,0]=m[i,0]+m[i+1+j,0]
                       m[i+1+j,0]=0                #We delete the
line we have just added
                       n=n+1
                   else:
                       n=n
    return delete_zeroes(m)

def real_part2(v):
    n=v.nrows()
    m=matrix(SR,2*v.nrows(),v.ncols())
    if v.ncols()==4:
        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]=v[i,2]
            m[i+n,2]=v[i,1]
            m[i+n,3]=v[i,3]
        n=0
        mlentgh=m.nrows()
        for i in range(mlentgh):
            if 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]) &
bool(m[i,2]==m[i+1+j,2]) & bool(m[i,3]==m[i+1+j,3]):
                        m[i,0]=m[i,0]+m[i+1+j,0]
                        m[i+1+j,0]=0                #We delete the
line we have just added
                        n=n+1
                    else:
                        n=n
    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,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]=v[i,3]
            m[i+n,3]=v[i,2]
            m[i+n,4]=v[i,4]
        n=0
        mlength=m.nrows()
        for i in range(mlength):
            if 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]) &
bool(m[i,2]==m[i+1+j,2]) & bool(m[i,3]==m[i+1+j,3]) &
bool(m[i,4]==m[i+1+j,4]):
                       m[i,0]=m[i,0]+m[i+1+j,0]
                       m[i+1+j,0]=0                #We delete the
line we have just added
                       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,4)
    if v.ncols()==4:
        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]
                m[i*w.nrows()+j,3]=v[i,3]+w[j,3]
        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]
                m[i*w.nrows()+j,3]=v[i,4]+w[j,4]
        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 v.ncols()==4:
        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
    else:
        for i in range(v.nrows()):
            m[i,0]=v[i,2]
            m[i,1]=v[i,3]
            m[i,2]=v[i,4]
            m[i,3]=v[i,0]
            m[i,4]=v[i,1]
        m=matrix(sorted(m))
        for i in range(v.nrows()):
            temp0=m[i,0]
            temp1=m[i,1]
            m[i,0]=m[i,3]
            m[i,1]=m[i,4]
            m[i,4]=m[i,2]
            m[i,2]=temp0
            m[i,3]=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,5)
    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]
            m[i*vector.nrows()+j,4]=scalar[i,3]+vector[j,4]
    return m
```

```
def bar(v):
    m=matrix(SR,v.nrows(),v.ncols())
    if v.ncols()==4:
        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]
            m[i,3]=v[i,3]
        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]
            m[i,4]=v[i,4]
        return m

def intz(v):
    length=0
    for i in range(v.nrows()):
        if bool((v[i,v.ncols()-3]+1).is_zero()): # if the
coefficient is z^{-1}\z^{b}\log^p|z|, the primitive has only one
components
            length=length+1
        else:
            length=length+v[i,v.ncols()-1]+1 # if the coefficient is
z^{a}\z^{b}\log^p|z| with a\neq -1, then the primitive has p+1
components
    m=matrix(SR,length,v.ncols())
    n=0
    if v.ncols()==4:
        for i in range(v.nrows()):
            if (v[i,1]+1).is_zero():        #integration of 1/z
                m[i+n,0]=2*v[i,0]/(v[i,3]+1)
                m[i+n,1]=0
                m[i+n,2]=v[i,2]
                m[i+n,3]=v[i,3]+1
            else:
                if v[i,3].is_zero():
                    m[i+n,0]=v[i,0]/(v[i,1]+1)
                    m[i+n,1]=v[i,1]+1
                    m[i+n,2]=v[i,2]
                else:
                    a0=1/(v[i,1]+1)
                    m[i+n,0]=a0*v[i,0]
                    m[i+n,1]=v[i,1]+1
                    m[i+n,2]=v[i,2]
                    m[i+n,3]=v[i,3]
                    n=n+1
                    for j in range(v[i,3]):
                        a0=-(v[i,3]-j)/(2*(v[i,1]+1))*a0
                        m[i+n,0]=a0*v[i,0]
```

```
                        m[i+n,1]=v[i,1]+1
                        m[i+n,2]=v[i,2]
                        m[i+n,3]=v[i,3]-j-1
                        n=n+1
                    n=n-1
        return m
    else:
        for i in range(v.nrows()):
            if (v[i,2]+1).is_zero():        #integration of 1/z
                m[i+n,0]=2*v[i,0]/(v[i,4]+1)
                m[i+n,1]=v[i,1]
                m[i+n,2]=0
                m[i+n,3]=v[i,3]
                m[i+n,4]=v[i,4]+1
            else:
                if v[i,4].is_zero():
                    m[i+n,0]=v[i,0]/(v[i,2]+1)
                    m[i+n,1]=v[i,1]
                    m[i+n,2]=v[i,2]+1
                    m[i+n,3]=v[i,3]
                else:
                    a0=1/(v[i,2]+1)
                    m[i+n,0]=a0*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]
                    m[i+n,4]=v[i,4]
                    n=n+1
                    for j in range(v[i,4]):
                        a0=-(v[i,4]-j)/(2*(v[i,2]+1))*a0
                        m[i+n,0]=a0*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]
                        m[i+n,4]=v[i,4]-j-1
                        n=n+1
                    n=n-1
        return m

def intzb(v):
    length=0
    for i in range(v.nrows()):
        if bool((v[i,v.ncols()-2]+1).is_zero()): # if the
coefficient is z^{-1}\z^{b}\log^p|z|, the primitive has only one
components
            length=length+1
        else:
            length=length+v[i,v.ncols()-1]+1 # if the coefficient is
z^{a}\z^{b}\log^p|z| with a\neq -1, then the primitive has p+1
components
    m=matrix(SR,length,v.ncols())
    n=0
    if v.ncols()==4:
        for i in range(v.nrows()):
```

```
        if (v[i,2]+1).is_zero():        #integration of 1/z
            m[i+n,0]=2*v[i,0]/(v[i,3]+1)
            m[i+n,1]=v[i,1]
            m[i+n,2]=0
            m[i+n,3]=v[i,3]+1
        else:
            if v[i,3].is_zero():
                m[i+n,0]=v[i,0]/(v[i,1]+1)
                m[i+n,1]=v[i,1]
                m[i+n,2]=v[i,2]+1
            else:
                a0=1/(v[i,2]+1)
                m[i+n,0]=a0*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]
                n=n+1
                for j in range(v[i,3]):
                    a0=-(v[i,3]-j)/(2*(v[i,2]+1))*a0
                    m[i+n,0]=a0*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]-j-1
                    n=n+1
                n=n-1
    return m
else:
    for i in range(v.nrows()):
        if (v[i,3]+1).is_zero():        #integration of 1/z
            m[i+n,0]=2*v[i,0]/(v[i,4]+1)
            m[i+n,1]=v[i,1]
            m[i+n,2]=v[i,2]
            m[i+n,3]=0
            m[i+n,4]=v[i,4]+1
        else:
            if v[i,4].is_zero():
                m[i+n,0]=v[i,0]/(v[i,3]+1)
                m[i+n,1]=v[i,1]
                m[i+n,2]=v[i,2]
                m[i+n,3]=v[i,3]+1
            else:
                a0=1/(v[i,3]+1)
                m[i+n,0]=a0*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
                m[i+n,4]=v[i,4]
                n=n+1
                for j in range(v[i,4]):
                    a0=-(v[i,4]-j)/(2*(v[i,3]+1))*a0
                    m[i+n,0]=a0*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
```

```
                        m[i+n,4]=v[i,4]-j-1
                        n=n+1
                    n=n-1
        return m

def diffz(v):
    length=v.nrows()
    if v.ncols()==4:
        for i in range(v.nrows()):
            if v[i,3]==0:
                length=length
            else:
                length=length+1
    else:
        for i in range(v.nrows()):
            if v[i,4]==0:
                length=length
            else:
                length=length+1
    m=matrix(SR,length,v.ncols())
    n=0
    if v.ncols()==4:
        for i in range(v.nrows()):
            if v[i,0].is_zero():
                n=n+1
            else:
                if bool(v[i,1]==0) & bool(v[i,3]==0):
                    n=n+1
                else:
                    if v[i,3]==0:
                        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:
                        if v[i,1]==0:
                            m[i-n,0]=v[i,3]*v[i,0]/2   # the factor
1/2 comes from the derivation of the log
                            m[i-n,1]=v[i,1]-1
                            m[i-n,2]=v[i,2]
                            m[i-n,3]=v[i,3]-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]
                            m[i-n,3]=v[i,3]
                            m[i-n+1,0]=v[i,3]*v[i,0]/2
                            m[i-n+1,1]=v[i,1]-1
                            m[i-n+1,2]=v[i,2]
                            m[i-n+1,3]=v[i,3]-1
                            n=n-1            # a we added another
term, we need to shift lines
    else:
        for i in range(v.nrows()):
            if v[i,0].is_zero():
```

```
                n=n+1
            else:
                if bool(v[i,2]==0) & bool(v[i,4]==0):
                    n=n+1
                else:
                    if v[i,4]==0:
                        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]
                    else:
                        if v[i,2]==0:
                            m[i-n,0]=v[i,4]*v[i,0]/2   # the factor
1/2 comes from the derivation of the log
                            m[i-n,1]=v[i,1]
                            m[i-n,2]=v[i,2]-1
                            m[i-n,3]=v[i,3]
                            m[i-n,4]=v[i,4]-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]
                            m[i-n,4]=v[i,4]
                            m[i-n+1,0]=v[i,4]*v[i,0]/2
                            m[i-n+1,1]=v[i,1]
                            m[i-n+1,2]=v[i,2]-1
                            m[i-n+1,3]=v[i,3]
                            m[i-n+1,4]=v[i,4]-1
                            n=n-1              # we added another
term, we need to shift lines
    return m.submatrix(0,0,v.nrows()-n,v.ncols())

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

```
                if bool(v[i,2]==0) & bool(v[i,3]==0):
                    n=n+1
                else:
                    if v[i,3]==0:
                        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:
                        if v[i,1]==0:
                            m[i-n,0]=v[i,3]*v[i,0]/2   # the factor
1/2 comes from the derivation of the log
                            m[i-n,1]=v[i,1]
                            m[i-n,2]=v[i,2]-1
                            m[i-n,3]=v[i,3]-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]
                            m[i-n+1,0]=v[i,3]*v[i,0]/2
                            m[i-n+1,1]=v[i,1]
                            m[i-n+1,2]=v[i,2]-1
                            m[i-n+1,3]=v[i,3]-1
                            n=n-1            # we added another
term, we need to shift lines
    else:
        for i in range(v.nrows()):
            if v[i,0].is_zero():
                n=n+1
            else:
                if bool(v[i,3]==0) & bool(v[i,4]==0):
                    n=n+1
                else:
                    if v[i,4]==0:
                        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
                    else:
                        if v[i,2]==0:
                            m[i-n,0]=v[i,4]*v[i,0]/2   # the factor
1/2 comes from the derivation of the log
                            m[i-n,1]=v[i,1]
                            m[i-n,2]=v[i,2]
                            m[i-n,3]=v[i,3]-1
                            m[i-n,4]=v[i,4]-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
                            m[i-n,4]=v[i,4]
                            m[i-n+1,0]=v[i,4]*v[i,0]/2
                            m[i-n+1,1]=v[i,1]
```

```
                            m[i-n+1,2]=v[i,2]
                            m[i-n+1,3]=v[i,3]-1
                            m[i-n+1,4]=v[i,4]-1
                            n=n-1             # we added another
term, we need to shift lines
    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 v.ncols()==4:
        for i in range(v.nrows()):
            if 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]
                m[i-n,3]=v[i,3]    #The logarithm do not matter for
throw, as we look at power of z up to |z|^{-\epsilon}
            else:
                n=n+1
    else:
        for i in range(v.nrows()):
            if 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]
                m[i-n,4]=v[i,4]
            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 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 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 j==v.ncols()-3:
                m[i,j]=v[i,j]-nz
            else:
                if j==v.ncols()-2:
                    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 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))

def scal_hold(v,w): #returns the scalar product of two vectors,
keeping track of the order of scalar products
    l = v.nrows()*w.nrows()
    m = matrix(SR,l,4)
    if v.ncols()==4:
        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]
                m[i*w.nrows()+j,3]=v[i,3]+w[j,3]
        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]
                m[i*w.nrows()+j,3]=v[i,4]+w[j,4]
        return m

#### \alpha_0=0 for all \theta_0\geq 4 thanks of the general proof


dzphi = matrix([[1,a0,3,0,0],[1,a1,4,0,0],[1,a2,5,0,0],
[1/16,c1,1,4,0],[1/8,c1b,3,2,0]])
g = matrix([[1,3,3,0],[2*abs(a1)^2,4,4,0],[alpha1,5,3,0],
[alpha1b,3,5,0]])
ginv = power_divide(taylor_inverse(power_divide(g,3,3),1,3),3,3)
H = matrix([[1/2,c1,-2,0,0],[1/2,c1b,0,-2,0],[1/2,c2,-1,0,0],
[1/2,c2b,0,-1,0],[1/2,b1,-2,1,0],[1/2,b1b,1,-2,0]])
h0 = Weingarten_simpler(dzphi,g,ginv,3,5)


latex(dzphi),latex(g),latex(ginv),latex(H),latex(h0)

Q=intQ(dzphi,ginv,H,h0,1)

latex(Q)

Htemp=matrix_sort(matrix_full_simplify(real_part(Q)))

latex(Htemp)

H2=sum_matrix(real_part2(matrix([[1,c1,-2,0,0],[1,c2,-1,0,0],
[1,c3,0,0,0],[1,b1,-2,1,0],[1,b2,-2,2,0],[1,b3,-1,1,0],
[1,e1,-4,4,0]])),matrix([1,gamma1,0,0,1]))

latex(H2)

dzphi2=sum_matrix(matrix([[1,a0,3,0,0],[1,a1,4,0,0],[1,a2,5,0,0],
[1,a3,6,0,0],
[1,a4,7,0,0]]),mult_scalar(1/2,intzb(throw(prod(g,H2),7))))

latex(dzphi2)

conf=factor_simplify(throw(scal(dzphi2,dzphi2),11))

g2temp=factor_simplify(mult_scalar(2,throw(scal(dzphi2,bar(dzphi2)),
11)))

latex(conf),latex(g2temp)

g2=sum_matrix(matrix([[1,3,3,0],[2*abs(a1)^2,4,4,0],
[beta,5,5,0]]),real_part2(matrix([[2*alpha1,5,3,0],[2*alpha2,1,8,0],
[2*alpha3,6,3,0],[2*alpha4,5,4,0],[2*alpha5,6,4,0],[2*alpha6,8,2,0],
[2*alpha7,9,1,0],[2*zeta0,7,3,1]])))
```

```
latex(g2)

g2inv=power_divide(taylor_inverse(power_divide(g2,3,3),2,5),3,3)

h02=Weingarten(dzphi2,g2,g2inv,4,7)

latex(g2inv),latex(h02)



e2u=power_divide(g2,3,3)

e2uinv=taylor_inverse(power_divide(g2,3,3),2,5)

Kh0square=Gauss_Weingarten(e2u,e2uinv,g2inv,h02,2)

Q=sing_quartic(g2inv,h02,2)

Qfinal=factor_simplify(sum_matrix(Q,Kh0square))

#mj(Qfinal)

#eq0=throw(diffz(diffzb(H2)),-1)
#eq1=factor_simplify(throw(diffzb(prod(scal(H2,H2),dzphi2)),-1))
#eq2=factor_simplify(throw(diffzb(mult_scalar(2,prod(scal(H2,h02),pr
od(g2inv,bar(dzphi2))))),-1))
#eq=factor_simplify(sum_matrix(eq0,real_part(sum_matrix(eq1,eq2))))

#latex(eq0),latex(eq1),latex(eq2),latex(eq)

#latex(g2),latex(g2inv),latex(h02),

latex(Kh0square),latex(Q),latex(Qfinal)


#### next order

Q=intQ(dzphi2,g2inv,H2,h02,2)

Htemp2=matrix_sort(matrix_full_simplify(real_part(Q)))

latex(Htemp2)


##### last order

H3=sum_matrix(real_part2(matrix([[1,c1,-2,0,0],[1,c2,-1,0,0],
[1,c3,0,0,0],[1,c4,1,0,0],[1,b1,-2,1,0],[1,b2,-2,2,0],[1,b4,-2,3,0],
[1,b3,-1,1,0],[1,b5,-1,2,0],[1,e1,-4,4,0],[1,e2,-4,5,0],
[1,e3,-3,4,0],[1,gamma2,1,0,1]])),matrix([1,gamma1,0,0,1]))

Q=intQ(dzphi2,g2inv,H3,h02,3)
```

```
Htemp=matrix_sort(matrix_full_simplify(real_part(Q)))

latex(H3),latex(Htemp)

H4=real_part2(matrix([[1,c1,-2,0,0],[1,c2,-1,0,0],[1,c3,0,0,0],
[1,c4,1,0,0],[1,c5,2,0,0],[1,c6,1,1,0],[1,b1,-2,1,0],[1,b2,-2,2,0],
[1,b4,-2,3,0],[1,b6,-2,4,0],[1,b3,-1,1,0],[1,b5,-1,2,0],
[1,b7,-1,3,0],[1,e1,-4,4,0],[1,e2,-4,5,0],[1,e4,-4,6,0],
[1,e3,-3,4,0],[1,e5,-3,5,0],[1,gamma1,0,0,1],[1,gamma2,1,0,1],
[1,gamma3,2,0,1],[1,gamma4,1,1,1]]))

latex(H4)

dzphi3=sum_matrix(matrix([[1,a0,3,0,0],[1,a1,4,0,0],[1,a2,5,0,0],
[1,a3,6,0,0],[1,a4,7,0,0],[1,a5,8,0,0],
[1,a6,9,0,0]]),mult_scalar(1/2,intzb(throw(prod(g2,H4),9))))

latex(dzphi3)

conf=matrix_full_simplify(throw(scal(dzphi3,dzphi3),13))

g3temp=matrix_full_simplify(mult_scalar(2,throw(scal(dzphi3,bar(dzph
i3)),13)))

latex(conf),latex(g3temp)


g3=sum_matrix(matrix([[1,3,3,0],[2*abs(a1)^2,4,4,0],[beta,5,5,0],
[delta,6,6,0]]),real_part2(matrix([[2*alpha1,5,3,0],
[2*alpha2,1,8,0],[2*alpha3,6,3,0],[2*alpha4,5,4,0],[2*alpha5,6,4,0],
[2*alpha6,8,2,0],[2*alpha7,9,1,0],[2*alpha8,10,1,0],
[2*alpha9,9,2,0],[2*alpha10,8,3,0],[2*alpha11,7,4,0],
[2*alpha12,6,5,0],[2*alpha13,13,-1,0],[2*alpha14,12,0,0],
[2*alpha15,11,1,0],[2*alpha16,10,2,0],[2*alpha17,9,3,0],
[2*alpha18,8,4,0],[2*alpha19,7,5,0],[2*zeta0,7,3,1],[2*zeta1,8,3,1],
[2*zeta2,9,3,1],[2*zeta3,8,4,1],[2*zeta4,7,5,1]])))

latex(g3)

g3inv=power_divide(taylor_inverse(power_divide(g3,3,3),3,7),3,3)

h03=Weingarten_simpler(dzphi3,g3,g3inv,6,9)

latex(g3),latex(g3inv),latex(matrix_sort(h03))

n=4

Q=sing_quartic(throw(g3inv,7-2*n),h03,4)  ## we do not need the full
g3, only the development up to order 5

latex(Q)


#### Gauss curvature
```

```
e2u=power_divide(g3,3,3)

e2uinv=taylor_inverse(power_divide(g3,3,3),3,7)

K=Gauss_Weingarten(e2u,e2uinv,g3inv,h03,4)

latex(K)

###

Q_end_end=matrix_full_simplify(sum_matrix(Q,K))

latex(Q_end_end)


### extra terms

H5=throw(H4,4-n) ### 4-n=0, we need two terms

h05=throw(h03,n+1)  ###idem

Hh0=throw(scal(H5,h05),3)

Hh0_square=factor_simplify(throw(scal_hold(Hh0,Hh0),4))

temp1=throw(scal(H5,H5),2)

temp2=throw(scal(h05,h05),8)

H2h02=mult_scalar(5/4,factor_simplify(throw(scal_hold(temp1,temp2),4
)))

unhold_Hh0_square=factor_simplify(throw(scal(Hh0,Hh0),4))

unhold_H2h02=mult_scalar(5/4,factor_simplify(throw(scal(temp1,temp2)
,4)))

latex(H2h02),latex(Hh0_square),latex(unhold_H2h02),latex(unhold_Hh0_
square)
```