Explicit computation of the height of a Gross-Schoen Cycle

Wang, R.

Chapter 5

Appendix

I Solving cubic equations

Given a cubic equation

$$ay^3 + by^2 + cy + d = 0,$$

we solve it in following way.

Algorithm 4 Solutions of a cubic equation

Input: a,b,c,d: coefficients of the polynomial
i: the index of expected solution, taking value in $\{1, 2, 3\}$
Output: the *i-th* solution of the polynomial

$r_3 = -\frac{1}{2} + \frac{\sqrt{3}}{2}i$
$u = \frac{9abc - 27a^2d - 2b^3}{54a^3}$
$v = \frac{(12ac^3 - 3b^2c^2 - 54abcd + 81a^2d^2 + 12b^3d)^{1/2}}{18a^2}$
if $|u + v| \geq |u - v|$ **then**
    $m = (u + v)^{(1/3)}$
else
    $m = (u - v)^{(1/3)}$
end if
if $m = 0$ **then**
    $n = 0$
else
    $n = \frac{b^2 - 3ac}{9am}$
end if
return $r_3^{i-1}m + r_3^{2i-2}n - \frac{b}{3a}$

When we say the $1st$ root of a cubic function, we mean the output of the algorithm above when $i = 1$, similarly for other indices i. The Magma code for our curve $\mathfrak{C}_{\mathbb{C}}$ is given as follows:

```
a:=1;
b:=x^2-x;
c:=-x^3;
d:=x^2+x;
cubic_unit:=-1/2+3^(1/2)/2*I;
root:=function(u,v,i);
numa:=a;
numb:=Evaluate(b,[u+v*I,0]);
numc:=Evaluate(c,[u+v*I,0]);
preuu:=(9*a*b*c-27*a^2*d-2*b^3)/(54*a^3);
uu:=Evaluate(preuu,[u+v*I,0]);
prevv:=Evaluate((3*(4*a*c^3-b^2*c^2-18*a*b*c*d+27*a^2*d^2+4*b^3*d)),[u+
    v*I,0]);
vv:=(prevv^(1/2))/(18*a^3);
m:=(uu-vv)^(1/3);
if Abs(uu+vv) ge Abs(uu-vv) then
m:=(uu+vv)^(1/3);
end if;
n:=0;
if m ne 0 then
n:=(numb^2-3*numa*numc)/(9*numa*m);
end if;
result:=(cubic_unit^(i-1))*m+n*cubic_unit^(2*i-2)-(numb)/(3*numa);
return result;
end function;
```

In Chapter 4, we need to integrate some functions defined on a plane quartic curve. It turns out that, for these integrations, we need to label the different y's for any fixed x in the way above.

II List of Weierstrass points

The set of Weierstrass points of $\mathfrak{C}_{\mathbb{C}}$ is:

$$
\begin{aligned}
&\mathfrak{W} = \{\\
&(0.0000000000, 0.0000000000),\\
&(-2.104587156, -0.3159811808),\\
&(-1.241974125, -1.824965133),\\
&(0.3539677313, -0.7304015296),\\
&(-1.959274470, -1.410535295),\\
&(-1.000000000, 0.0000000000),\\
&(-1.455757161 - 1.277524807 * I, -0.5258271579 - 3.390009274 * I),\\
&(-1.455757161 + 1.277524807 * I, -0.5258271579 + 3.390009274 * I),\\
&(1.266455202 - 1.314861880 * I, 0.1352071928 + 2.966892635 * I),\\
&(1.266455202 + 1.314861880 * I, 0.1352071928 - 2.966892635 * I),\\
&(0.2089717154 - 1.597843531 * I, 0.1450907026 - 2.034168656 * I),\\
&(0.2089717154 + 1.597843531 * I, 0.1450907026 + 2.034168656 * I),\\
&(-1.319681529 - 0.4985373090 * I, -0.4698939284 - 0.6348619710 * I),\\
&(-1.319681529 + 0.4985373090 * I, -0.4698939284 + 0.6348619710 * I),\\
&(1.691125343 - 0.6842628628 * I, 0.6312556990 + 0.6046227212 * I),\\
&(1.691125343 + 0.6842628628 * I, 0.6312556990 - 0.6046227212 * I),\\
&(-1.125631934 - 0.2207052168 * I, -0.6596966755 + 0.07228789093 * I),\\
&(-1.125631934 + 0.2207052168 * I, -0.6596966755 - 0.07228789093 * I),\\
&(-0.5282412488 - 1.005235134 * I, -0.4177679369 + 0.3210649478 * I),\\
&(-0.5282412488 + 1.005235134 * I, -0.4177679369 - 0.3210649478 * I),\\
&(-0.3349732907 - 0.4139797966 * I, -0.4206274002 - 0.9204898342 * I),\\
&(-0.3349732907 + 0.4139797966 * I, -0.4206274002 + 0.9204898342 * I),\\
&(0.07366689191 - 0.3721976724 * I, 0.7232010036 + 0.2545999915 * I),\\
&(0.07366689191 + 0.3721976724 * I, 0.7232010036 - 0.2545999915 * I)\}
\end{aligned}
$$

III Code for canonical form

```
from sage.schemes.riemann_surfaces.riemann_surface import
    RiemannSurface
R.<x,y> = QQ[]
f = -x^3*y+x^2*y^2-x*y^2+y^3+x^2+x

S = RiemannSurface(f)
M = S.period_matrix()
Omega1 = M[[0,1,2],[0,1,2]]
Omega2 = M[[0,1,2],[3,4,5]]
A = Omega1.apply_map(real)
B = Omega1.apply_map(imag)
tau = S.riemann_matrix()
Omega1bar= A-B*I
Imtau = tau.apply_map(imag)
RieSecBil = Omega1bar*Imtau*Omega1.transpose()
RieSecBil
[ 8.14058999836226 - 4.44089209850063e-16*I -2.71111012021317 -
    9.21762666195036e-14*I -5.21995559097122 - 5.85642645489770e-14*I
    ]
[-2.71111012021315 + 6.52811138479592e-14*I  8.66278158093292 +
    8.32667268468867e-14*I  5.65493670273040 + 1.56541446472147e-13*I
    ]
[-5.21995559097118 + 1.45994327738208e-14*I  5.65493670273037 -
    1.75831571525009e-14*I  10.7557438794561 + 1.18932641512970e-14*I
    ]

w11 = RieSecBil[0][0]
w1y = RieSecBil[0][1]
w1x = RieSecBil[0][2]
wyy = RieSecBil[1][1]
wyx = RieSecBil[1][2]
wxx = RieSecBil[2][2]
k = (wyx-w1x*w1y/w11)/(wyy-w1y*w1y/w11)
norm1 = RieSecBil[0][0]**(1/2)
norm2 = (wyy-(2*w1y*w1y/w11)+w1y*w1y/w11)**(1/2)
norm3 = (wxx+w1x*w1x/w11+k*k*wyy+k*k*w1y*w1y/w11-2*w1x*w1x/w11-2*k*
    wyx+2*k*w1y*w1x/w11+2*k*w1x*w1y/w11-2*k*w1x*w1y/w11-2*k*k*w1y*w1y
    /w11)**(1/2)
```

Gram-Schmidt process

In the code, the symbol wmn denotes the integration of 1-1 forms, for example, $w1y$ represents $\frac{i}{2}\int_X \omega_1 \wedge \overline{\omega}_y$. The Gram-Schmidt process is actually done by hand, and we just use SageMath for evaluation.

IV Code for semistability

```
from collections import defaultdict
PP.<x,y,z> = ProjectiveSpace(QQ,2)
Poly.<U,V> = PolynomialRing(ZZ, 2)
RR.<X, Y, Z> = PolynomialRing(ZZ, 3)
C = Curve(-x^3*y+x^2*y^2-x*y^2*z+y^3*z+x^2*z^2+x*z^3, PP)
inj1 = Poly.hom([X, Y], RR)


def IsDegree1(I):    #checking the ideal I has degree 1 generators
out = True
for f in I.gens():
if not f.degree() == 1:
out = False
return out


def MyBadPrimes(C):    #finding out primes with bad reduction
f = C.defining_polynomial()
RZ.<xZ,yZ,zZ> = PolynomialRing(ZZ, 3)
coeffs = f.coefficients()
dens = [c.denominator() for c in coeffs]
den = lcm(dens)
F = RZ(f*den)
Fx = F.derivative(xZ)
Fy = F.derivative(yZ)
Fz = F.derivative(zZ)
NaiveDisc = 1
for P in [[xZ,yZ,1],[xZ,1,zZ],[1,yZ,zZ]]:
I = ideal([g(P) for g in [F,Fx,Fy,Fz]])
G = I.groebner_basis()
n = G[len(G)-1]
NaiveDisc = lcm(n, NaiveDisc)
return [a[0] for a in factor(NaiveDisc)]


def MyBadPoints(C, default_prime):    #CORE FUNCTION: finding out
    all singular points over certain prime number and the tangent
    lines
f = C.defining_polynomial()
RZ.<xZ,yZ,zZ> = PolynomialRing(ZZ, 3)
coeffs = f.coefficients()
dens = [c.denominator() for c in coeffs]
den = lcm(dens)
F = RZ(f*den)
```

```
        Fx = F.derivative(xZ)
        Fy = F.derivative(yZ)
        Fz = F.derivative(zZ)
        Out = ['Dimention of singular locus is positive.']
        GG = []                #list of associate primes (after field
            extension)
        sing_set = []          #list of degree 2 parts of singular points
            after transition
        factor_sing = []       #list of factorizations of degree 2 parts
        degree = 1
        SingDim = 0
        found = False
        while found == False:
        Rp = RZ.change_ring(GF(default_prime^degree))
        win = True
        S.<u, v> = PolynomialRing(GF(default_prime^degree), 2)
        for P in [ [u, v, 1], [u, 1, v], [1, u, v] ]:
        affine_patch_map = RZ.hom(P, S)
        I_2 = Ideal(S,[ affine_patch_map(g) for g in [F, Fx, Fy, Fz]])
        G_2 = I_2.associated_primes()
        for I in G_2:
        VV = I.gens()
        if Set(VV).cardinality()==1:
        SingDim = 1
        if SingDim == 1:
        break
        for I in G_2:
        if IsDegree1(I) == False:
        win = False          #This means that the singular points don't have
            coordinates in this base field and we still need to extend it
        if SingDim == 1:
        break
        Out = []
        if win == True:
        for P in [ [u, v, 1], [u, 1, v], [1, u, v] ]:
        affine_patch_map = RZ.hom(P, S)
        I_2 = Ideal(S,[affine_patch_map(g) for g in [F, Fx, Fy, Fz]])
        G_2 = I_2.associated_primes()
        GG += G_2
        for AP in G_2:          #in this for loop, we find out the coordinate
             of singular points
        T.<x, y> = S.quotient(ideal(S, [w for w in AP.gens()]))
        Quot = T.cover()
        if Quot(1) == 0:
        continue
        for m in GF(default_prime^degree):
        if Quot(u-m)==0:
```

```
        SingX = m
87      break
        for n in GF(default_prime^degree):
89      if Quot(v-n)==0:
        SingY = n
91      break
        Trans = S.hom([u+SingX, v+SingY])
93      change_to_S = RZ.hom(P, S)
        TranP = Trans(change_to_S(F))    #our curve after transition
95      homog_part = defaultdict(TranP.parent())    #the following 4 lines
            help us to get the degree 2 part
        for coeff,monom in TranP:
97      homog_part[monom.degree()] += coeff * monom
        sing_set.append(homog_part[2])
99      found = True
        Out += GG
101     for w in sing_set:    #find out the singularity type
        SS = S.change_ring(GF(default_prime^(2*degree)))
103     if w!=0:
        factor_sing.append(factor(SS(w)))
105     else:
        factor_sing.append('HIGHER_SINGULARITY')
107     else: degree += 1
        return Out, factor_sing
```

V Code for semi-canonical divisor

In Magma, a base point P_{bs} is already chosen and $AbelJacobi(D)$ means $AJ(D-\deg(D)[P_{bs}])$.

```
PeriodMatrix:=Matrix([
[-0.075648772682765 8975 + 0.8285079351867007 0913*I,
0.4865658558914058 6329 - 0.0753194443604481 4614*I,
0.5323850494490135 0223 + 0.2314338434720501 6268*I],
[0.4865658558914058 6329 - 0.0753194443604481 4614*I,
0.5005849211118226 23585 + 0.4887408003732397 4253*I,
0.1378801009244352 44693 - 0.0570373001175851 4006*I],
[0.5323850494490135 0223 + 0.2314338434720501 6268*I,
0.1378801009244352 44693 - 0.0570373001175851 4006*I,
0.6897178783724336 09278 + 0.6926477435047031 3589*I]])
```

```
C<I> := ComplexField(30);
Qxy<x, y> := PolynomialRing(Rationals(), 2);
f := -x^3*y+x^2*y^2-x*y^2+y^3+x^2+x;     //defining polynomial
X := RiemannSurface(f);     //create the riemann surface
tau:=SmallPeriodMatrix(X);

Pt := X ! [0,0];     // a point on X
Div:=Divisor([Pt],[2]);     //effective divisor of degree 2
Div2:=Divisor([BasePoint(X)],[2]);
Div3:=Divisor([BasePoint(X),Pt],[1,1]);

asd := InfinitePoints(X);
U:=[asd[1],asd[2],asd[3]];
V:=[2,1,1];
CanDiv:=Divisor(U,V);     //canonical divisor

MatrixIm := function(M)     //function to take the imaginary part of a
     matrix.
N := ZeroMatrix(C, Nrows(M), Ncols(M));
for i in [1..Nrows(M)] do
for j in [1..Ncols(M)] do
N[i,j] := Imaginary(M[i,j]);
end for;
end for;
return N;
end function;

Columns := function(M)     //function to extract columns as a list of
    matrices
```

```
28      out := [];
        for i in [1..Ncols(M)] do
30      out := out cat [ColumnSubmatrix(M, i, 1)];
        end for;
32      return out;
        end function;
34
        char:=Matrix([[0],[0],[0],[0],[0],[0]]);
36      Imtau:=MatrixIm(tau);
        detimtau:=Determinant(Imtau);
38      LeftPeriod:=RemoveColumn(RemoveColumn(RemoveColumn(BigPeriodMatrix(X)
            ,4),4),4);
        ID3 := DiagonalMatrix(C, [1,1,1]);
40      columnlist := Columns(ID3) cat Columns(tau);
        AJCD := Matrix(AbelJacobi(CanDiv));     //the image of canonical
            divisor under AJ map
42
        AJ_2_torsion := [];
44      for v in CartesianPower([0,1], 6) do
        new := ZeroMatrix(C, 3, 1);
46      for i in [1..6] do
        new := new + v[i] * columnlist[i];
48      end for;
        AJ_2_torsion := AJ_2_torsion cat [(1/2) * new];
50      end for;

52      translates_to_try := [];     //this will be a list of all 64 square
            roots of CanDiv
        for V in AJ_2_torsion do
54      translates_to_try := translates_to_try cat [(1/2) * AJCD + V];
        end for;
56
        torus_theta_function := function(D)// D is a divisor of degree g-1
58      myinput := Matrix(AbelJacobi(D));
        myabstheta:=Abs(Theta(char, myinput, tau));
60      myiminput:= MatrixIm(myinput);
        myexponent:=-3.141592653589*Transpose(myiminput)*(Imtau^(-1))*
            myiminput;
62      return Exp(myexponent[1,1])*myabstheta*detimtau^(1/4);
        end function;
64
        find_correct_theta_translate := function(D)     //find out the correct
             translation
66      P := AbelJacobi(D);
        out := [];
68      for V in translates_to_try do
        myinput := Matrix(P) - V;
```

```
70      myabstheta:=Abs(Theta(char, myinput, tau));
        myiminput:= MatrixIm(myinput);
72      myexponent:=-3.141592653589*Transpose(myiminput)*(Imtau^(-1))*
            myiminput;
        if Abs(Exp(myexponent[1,1])*myabstheta*detimtau^(1/4)) le 0.00000001
            then
74      return V;
        end if;
76      end for;
        end function;
78
        //we check the result by choosing 3 different effective divisors
80      ThetaTranslate1 := find_correct_theta_translate(Div);
        ThetaTranslate1;
82      ThetaTranslate2 := find_correct_theta_translate(Div2);
        ThetaTranslate2;
84      ThetaTranslate3 := find_correct_theta_translate(Div3);
        ThetaTranslate3;
```

VI Code for $\|\chi'_{18}\|_{\mathbf{Hdg}}$

```
C<I> := ComplexField(50);
Qxy<x, y> := PolynomialRing(Rationals(), 2);
f := -x^3*y+x^2*y^2-x*y^2+y^3+x^2+x;     //defining polynomial
X := RiemannSurface(f);     //create the riemann surface

tau1:=SmallPeriodMatrix(X);     //period matrix implemented in Magma
tau2:=Matrix(C,3,[  0.855638485763810 + 1.03033263075936*I,
    0.380495294693576 + 0.00436338826915927*I,   0.506949568788359 -
    0.178115589372895*I,
0.380495294693584 + 0.00436338826915617*I ,  0.401072147476828 +
    0.694848752465586*I , -0.580339267402228 - 0.147446853346675*I,
0.506949568788374 - 0.178115589372884*I,  -0.580339267402254 -
    0.147446853346687*I,   0.219761917471621 + 0.625482641303850*I
]);     //period matrix implemented in SageMath

//the following set S contains all the even characteristics
S:=[Matrix(RationalField(),1,[0,0,0,0,0,0]),Matrix(RationalField()
    ,1,[0,0,0,0,0,1]),Matrix(RationalField(),1,[0,0,0,0,1,0]),Matrix(
    RationalField(),1,[0,0,0,0,1,1]),Matrix(RationalField()
    ,1,[0,0,0,1,0,0]),Matrix(RationalField(),1,[0,0,0,1,0,1]),Matrix(
    RationalField(),1,[0,0,0,1,1,0]),Matrix(RationalField()
    ,1,[0,0,0,1,1,1]),Matrix(RationalField(),1,[0,0,1,0,0,0]),Matrix(
    RationalField(),1,[0,0,1,0,1,0]),Matrix(RationalField()
    ,1,[0,0,1,1,0,0]),Matrix(RationalField(),1,[0,0,1,1,1,0]),Matrix(
    RationalField(),1,[0,1,0,0,0,0]),Matrix(RationalField()
    ,1,[0,1,0,0,0,1]),Matrix(RationalField(),1,[0,1,0,1,0,0]),Matrix(
    RationalField(),1,[0,1,0,1,0,1]),Matrix(RationalField()
    ,1,[0,1,1,0,0,0]),Matrix(RationalField(),1,[0,1,1,1,0,0]),Matrix(
    RationalField(),1,[0,1,1,1,1,1]),Matrix(RationalField()
    ,1,[0,1,1,0,1,1]),Matrix(RationalField(),1,[1,0,0,0,0,0]),Matrix(
    RationalField(),1,[1,0,0,0,0,1]),Matrix(RationalField()
    ,1,[1,0,0,0,1,0]),Matrix(RationalField(),1,[1,0,0,0,1,1]),Matrix(
    RationalField(),1,[1,0,1,0,0,0]),Matrix(RationalField()
    ,1,[1,0,1,0,1,0]),Matrix(RationalField(),1,[1,0,1,1,0,1]),Matrix(
    RationalField(),1,[1,0,1,1,1,1]),Matrix(RationalField()
    ,1,[1,1,0,0,0,0]),Matrix(RationalField(),1,[1,1,0,0,0,1]),Matrix(
    RationalField(),1,[1,1,0,1,1,0]),Matrix(RationalField()
    ,1,[1,1,0,1,1,1]),Matrix(RationalField(),1,[1,1,1,0,0,0]),Matrix(
    RationalField(),1,[1,1,1,0,1,1]),Matrix(RationalField()
    ,1,[1,1,1,1,1,0]),Matrix(RationalField(),1,[1,1,1,1,0,1])];

MatrixIm := function(M)     //imaginary part of a matrix
N := ZeroMatrix(C, Nrows(M), Ncols(M));
for i in [1..Nrows(M)] do
```

```
        for j in [1..Ncols(M)] do
19      N[i,j] := Imaginary(M[i,j]);
        end for;
21      end for;
        return N;
23      end function;

25      compute:=function(period)      //compute ||theta||
        chitilde:=1;
27      zerovector:=Matrix(C,1,[0,0,0]);
        for ele in S do
29      chitilde:=chitilde*Theta((1/2)*ele,zerovector,period);
        end for;
31      abschitilde:=Abs(chitilde);
        Imtau:=MatrixIm(period);
33      det:=Determinant(Imtau);
        prepreresult:=Log((det^9)*abschitilde);
35      preresult:=26*Log(2)+54*Log(Pi(C))+prepreresult;
        result:=-(21/18)*preresult;     //contribution of \chi_18 at the
            infinite place
37      return result;
        end function;
39
        //we compute the result using two period matrices implemented in
            Magma and SageMath
41      compute(tau1);
        compute(tau2);
```

VII Code for the integration part in $\log(S(\mathfrak{C}_{\mathbb{C}}))$

```
C<I> := ComplexField(30);
Cxy<x, y> := PolynomialRing(RationalField(), 2);
f := -x^3*y+x^2*y^2-x*y^2+y^3+x^2+x;      //defining polynomial
X := RiemannSurface(f : Precision:=30);     //create riemann surface
tau:=SmallPeriodMatrix(X);
BP:=BasePoint(X);

MatrixIm := function(M)     //take the imaginary part of a matrix
N := ZeroMatrix(C, Nrows(M), Ncols(M));
for i in [1..Nrows(M)] do
for j in [1..Ncols(M)] do
N[i,j] := Imaginary(M[i,j]);
end for;
end for;
return N;
end function;

Imtau:=MatrixIm(tau);
detimtau:=Determinant(Imtau);
ThetaTranslate:=Matrix([[0.47925054265168018675828130077 -
    0.00334176833187451614116524746349*I],
[0.69868487750843232229093518084 1 + 0.19949572388256356309872784152 5*
    I],
[0.00722266620787249384535143960054 -
    0.04301020693432081496232504368 05*I]]);

normalizedtheta := function(D);     //compute ||theta||_{g-1} at D
P := AbelJacobi(D);
myinput := Matrix(P) - ThetaTranslate;
char:=Matrix([[0],[0],[0],[0],[0],[0]]);
myabstheta:=Abs(Theta(char, myinput, tau));
myiminput:= MatrixIm(myinput);
myexponent:=-3.141592653589*Transpose(myiminput)*(Imtau^-1)*myiminput
    ;
return Abs(Exp(myexponent[1,1])*myabstheta*detimtau^(1/4));
end function;

a:=1;
b:=x^2-x;
c:=-x^3;
d:=x^2+x;
cubic_unit:=-1/2+3^(1/2)/2*I;
```

```
40      root:=function(u,v,i);     //solve the cubic equation with
            coefficients a, b, c and d.
        numa:=a;
42      numb:=Evaluate(b,[u+v*I,0]);
        numc:=Evaluate(c,[u+v*I,0]);
44      preuu:=(9*a*b*c-27*a^2*d-2*b^3)/(54*a^3);
        uu:=Evaluate(preuu,[u+v*I,0]);
46      prevv:=Evaluate((3*(4*a*c^3-b^2*c^2-18*a*b*c*d+27*a^2*d^2+4*b^3*d)),[
            u+v*I,0]);
        vv:=(prevv^(1/2))/(18*a^2);
48      m:=(uu-vv)^(1/3);
        if Abs(uu+vv) ge Abs(uu-vv) then
50      m:=(uu+vv)^(1/3);
        end if;
52      n:=0;
        if (Abs(m) ge 0.00001) then
54      n:=(numb^2-3*numa*numc)/(9*numa*m);
        else n:=0;
56      end if;
        result:=(cubic_unit^(i-1))*m+n*cubic_unit^(2*i-2)-(numb)/(3*numa);
58      return result;
        end function;
60
        embedding:=function(u,v,i);     //find out the embedding index in
            Magma
62      zz,pt1:=IsPoint(X,<[u,v],1>);
        zz,pt2:=IsPoint(X,<[u,v],2>);
64      zz,pt3:=IsPoint(X,<[u,v],3>);
        if (Abs(Coordinates(pt1)[2]-root(u,v,i)) le 0.0001) then result:=1;
66      elif (Abs(Coordinates(pt2)[2]-root(u,v,i)) le 0.0001) then result:=2;
        elif (Abs(Coordinates(pt3)[2]-root(u,v,i)) le 0.0001) then result:=3;
68      end if;
        return result;
70      end function;

72      theta_wrt_yi:=function(u,v,i);     //for a fixed point BP, evaluate
            theta_{g-1}(gP-Q) at the point Q=(Rex=u, Imx=v, index=i)
        zz, pt_in_cover_i:=IsPoint(X,<[u,v],embedding(u,v,i)>);
74      targetdivisor:=Divisor([ BP,pt_in_cover_i],[3,-1]);
        return normalizedtheta(targetdivisor);
76      end function;

78      omega1 := function(u,v,i)     // first differential form, i denotes
            index of y
        u:=u+0.000001;
80      v:=v+0.000001;
        x:=u+v*I;
```

```
82      y:=root(u,v,i);
        return 1/(-x^3+2*x^2*y-2*x*y+3*y^2);
84      end function;

86      omegax := function(u,v,i)     //second differential form, i denotes
            index of y
        u:=u+0.000001;
88      v:=v+0.000001;
        x:=u+v*I;
90      y:=root(u,v,i);
        return x/(-x^3+2*x^2*y-2*x*y+3*y^2);
92      end function;

94      omegay := function(u,v,i)      //third differential form, i denotes
            index of y
        u:=u+0.000001;
96      v:=v+0.000001;
        x:=u+v*I;
98      y:=root(u,v,i);
        return y/(-x^3+2*x^2*y-2*x*y+3*y^2);
100     end function;

102     canonicalform:=function(u,v,i)     //the volume form,i denotes index
            of y
        ort_nor_w1:=0.350487116953118*omega1(u,v,i);
104     ort_nor_wy:=(0.358981759779085*omegay(u,v,i)+0.119553875346235*omega1
            (u,v,i));
        ort_nor_wx:=(0.203008239643111*omega1(u,v,i)-0.216555180015011*omegay
            (u,v,i)+0.429067210690657*omegax(u,v,i));
106     result:=(ort_nor_w1*Conjugate(ort_nor_w1)+ort_nor_wy*Conjugate(
            ort_nor_wy)+ort_nor_wx*Conjugate(ort_nor_wx))/3;
        return result;
108     end function;

110     ff:=function(u,v)     //compute three i in one time
        result:= Log(theta_wrt_yi(u,v,1))*(canonicalform(u,v,1))+Log(
            theta_wrt_yi(u,v,2))*(canonicalform(u,v,2))+Log(theta_wrt_yi(u,v
            ,3))*(canonicalform(u,v,3));
112     return result;
        end function;
114
        SX:=0;     //take Riemann sum
116     scale:=0.01;
        rex_start:=-10;
118     imx_start:=-10;
        for p in [1..200] do
120     for q in [1..2000] do
```

```
        SX:=SX+ff(p*scale+rex_start+0.005,q*scale+imx_start+0.005)*scale*
            scale*(-9);
122     end for;
        end for;
124     SXn10:=SX;
        SXn10;
```

We can change the *rex_start* and *imx_start* in the code to get Riemann sums for a selected area. The edge length of small squares is *scale*. We choose $scale = 0.01$ when $-10 \leq x \leq 10$ and $-10 \leq y \leq 10$, and choose $scale = 0.1$ for other region in $-50 \leq x \leq 50$ and $-50 \leq y \leq 50$.

In this code, we only compute the Riemann sums for points with coordinates in $-10 \leq x \leq -8$ and $-10 \leq y \leq 10$ (we choose $scale = 0.01$). We need to change *rex_start* and *imx_start* manually to get the Riemann sums of other region.

The reason of doing this is that we can split the computation into small pieces, so that we can parallelly compute them in Magma. This can reduce the computing time significantly. The codes above takes around 40 hours. If we carry out all the computation ($-50 \leq x \leq 50$ and $-50 \leq y \leq 50$) in one time, it will take more than 600 hours. We also use this trick in the computation of $H(\mathfrak{C}_{\mathbb{C}})$.

VIII Code for $T(X)$

```
C<I> := ComplexField(50);     //we need high precision here, since
    components of T(X) can be very small
Cxy<x, y> := PolynomialRing(RationalField(), 2);
f := -x^3*y+x^2*y^2-x*y^2+y^3+x^2+x;     //defining polynomial
X := RiemannSurface(f : Precision:=15);     //create riemann surface
tau:=SmallPeriodMatrix(X);

MatrixIm := function(M)     //function to take the imaginary part of a
     matrix with entries in the field C defined just above.
N := ZeroMatrix(C, Nrows(M), Ncols(M));
for i in [1..Nrows(M)] do
for j in [1..Ncols(M)] do
N[i,j] := Imaginary(M[i,j]);
end for;
end for;
return N;
end function;

Imtau:=MatrixIm(tau);
detimtau:=Determinant(Imtau);
ThetaTranslate:=Matrix([[0.47925054265168018675828130077 4 -
    0.00334176833187451614116524746349*I],
[0.69868487750843232229093518084 1 + 0.19949572388256356309872784152 5*
    I],
[0.00722266620787249384535143960054 -
    0.04301020693432081496232504368 05*I]]);

normalizedtheta := function(D);     // ||theta||_{g-1}
P := AbelJacobi(D);
myinput := Matrix(P) - ThetaTranslate;
char:=Matrix([[0],[0],[0],[0],[0],[0]]);
myabstheta:=Abs(Theta(char, myinput, tau));
myiminput:= MatrixIm(myinput);
myexponent:=-3.141592653589*Transpose(myiminput)*(Imtau^-1)*myiminput
    ;
return Abs(Exp(myexponent[1,1])*myabstheta*detimtau^(1/4));
end function;

a:=1;     //the following lines are the coefficient of the defining
    polynomial in y
b:=x^2-x;
c:=-x^3;
d:=x^2+x;
cubic_unit:=-1/2+3^(1/2)/2*I;
```

```
39      root:=function(u,v,i);     //solve equation with coefficients a,b,c,d
        numa:=a;
41      numb:=Evaluate(b,[u+v*I,0]);
        numc:=Evaluate(c,[u+v*I,0]);
43      preuu:=(9*a*b*c-27*a^2*d-2*b^3)/(54*a^3);
        uu:=Evaluate(preuu,[u+v*I,0]);
45      prevv:=Evaluate((3*(4*a*c^3-b^2*c^2-18*a*b*c*d+27*a^2*d^2+4*b^3*d)),[
            u+v*I,0]);
        vv:=(prevv^(1/2))/(18*a^2);
47      m:=(uu-vv)^(1/3);
        if Abs(uu+vv) ge Abs(uu-vv) then
49      m:=(uu+vv)^(1/3);
        end if;
51      n:=0;
        if m ne 0 then
53      n:=(numb^2-3*numa*numc)/(9*numa*m);
        end if;
55      result:=(cubic_unit^(i-1))*m+n*cubic_unit^(2*i-2)-(numb)/(3*numa);
        return result;
57      end function;

59      wronskisquare:=function(x,y)
        fy:=-x^3+2*x^2*y-2*x*y+3*y^2;     //partial derivative of defining
            polynomial wrt y
61      dy:=(3*x^2*y-2*x*y^2+y^2-2*x-1)/(-x^3+2*x^2*y-2*x*y+3*y^2);     //
            implicit derivative of y wrt x
        ddy:=(6*x*y+3*x^2*dy+3*x^2*dy-2*y^2-4*x*y*dy-4*x*y*dy-2*x^2*dy^2+2*y*
            dy+2*y*dy+2*x*dy^2-6*y*dy^2-2)/(-x^3+2*x^2*y-2*x*y+3*y^2);     //
            second derivative
63      dfy:=-3*x^2+4*x*y+2*x^2*dy-2*y-2*x*dy+6*y*dy;     //implicit
            derivative of fy wrt x
        ddfy:=-6*x+4*y+4*x*dy+4*x*dy+2*x^2*ddy-2*dy-2*dy-2*x*ddy+6*dy^2+6*y*
            ddy;     //second derivative
65      w1:=1/fy;     //differential forms
        wy:=y/fy;
67      wx:=x/fy;
        dw1:=-dfy/fy^2;     //implicit derivative of w1 wrt x
69      dwy:=(dy*fy-dfy*y)/(fy^2);
        dwx:=(fy-x*dfy)/(fy^2);
71      ddw1:=-(ddfy*fy^2-2*fy*dfy^2)/(fy^4);     //second derivative
        ddwy:=((ddy*fy+dy*dfy-ddfy*y-dfy*dy)*fy^2-2*fy*dfy*(dy*fy-dfy*y))/(fy
            ^4);
73      ddwx:=((dfy-dfy-x*ddfy)*fy^2-2*fy*dfy*(fy-x*dfy))/(fy^4);
        c11:=0.350487116953118;     //orthogonalizing matrix
75      c21:=0.119553875346235;
        c22:=0.358981759779085;
```

```
77      c31:=0.203008239643111;
        c32:=-0.216555180015011;
79      c33:=0.429067210690657;
        W11:=c11*w1;     //following lines gives the wronskian matrix wrt a
            orthonormal basis of differential form.
81      W12:=c21*w1+c22*wy;
        W13:=c31*w1+c32*wy+c33*wx;
83      W21:=c11*dw1;
        W22:=c21*dw1+c22*dwy;
85      W23:=c31*dw1+c32*dwy+c33*dwx;
        W31:=c11*ddw1;
87      W32:=c21*ddw1+c22*ddwy;
        W33:=c31*ddw1+c32*ddwy+c33*ddwx;
89      Wronski:=Matrix([[W11,W12,W13],[W21,W22,W23],[W31/2,W32/2,W33/2]]);
        detwro:=Abs(Determinant(Wronski));
91      return detwro^2;
        end function;
93
        embedding:=function(u,v,i);     //this gives the embedding index of
            point (u+v*I,y_i) on \cC  implemented in Magma
95      zz,pt1:=IsPoint(X,<[u,v],1>);
        zz,pt2:=IsPoint(X,<[u,v],2>);
97      zz,pt3:=IsPoint(X,<[u,v],3>);
        if (Abs(Coordinates(pt1)[2]-root(u,v,i)) le 0.01) then result:=1;
99      elif (Abs(Coordinates(pt2)[2]-root(u,v,i)) le 0.01) then result:=2;
        elif (Abs(Coordinates(pt3)[2]-root(u,v,i)) le 0.01) then result:=3;
101     end if;
        return result;
103     end function;

105     WeierPts_coor:=[[ 0, 0], [ -2.104587155963303, -0.3159811807558051],
            [ (-1.455757161115023 - 1.277524806650578*I),
            (-0.5258271579143416 - 3.390009274518869*I)], [
            (-1.455757161115023 + 1.277524806650578*I), (-0.5258271579143403
            + 3.390009274518867*I)], [ -1.95927446954141,
            -1.410535295372148], [ (1.266455202041324 - 1.31486188027894*I),
            (0.1352071928093522 + 2.966892634578606*I)], [ (1.266455202041324
             + 1.31486188027894*I), (0.1352071928093521 - 2.966892634578606*I
            )], [ (0.2089717154013823 - 1.59784353074737*I),
            (0.1450907026008794 - 2.034168656250672*I)], [
            (0.2089717154013823 + 1.59784353074737*I), (0.1450907026008798 +
            2.034168656250671*I)], [ (-1.31968152920268 - 0.4985373089678906*
            I), (-0.4698939283630282 - 0.6348619710424462*I)], [
            (-1.31968152920268 + 0.4985373089678906*I), (-0.4698939283630282
            + 0.6348619710424618*I)], [ (1.691125343336006 -
            0.6842628628388769*I), (0.6312556990319607 + 0.6046227212351708*I
            )], [ (1.691125343336006 + 0.6842628628388769*I),
```

```
        (0.6312556990319608 - 0.6046227212351708*I)], [
        (-1.125631933711411 - 0.2207052167744392*I), (-0.6596966755302974
         + 0.07228789092962729*I)], [ (-1.125631933711411 +
        0.2207052167744392*I), (-0.6596966755302974 -
        0.07228789092962723*I)], [ (-0.5282412487764151 -
        1.005235133687909*I), (-0.4177679368816311 + 0.3210649477940537*I
        )], [ (-0.5282412487764151 + 1.005235133687909*I),
        (-0.4177679368816311 - 0.3210649477940537*I)], [
        -1.241974125539051, -1.824965132496513], [ 0.3539677313137965,
        -0.7304015296367112], [ (-0.3349732907033383 -
        0.4139797966480703*I), (-0.4206274001987688 - 0.9204898341402288*
        I)], [ (-0.3349732907033383 + 0.4139797966480703*I),
        (-0.4206274001987688 + 0.9204898341402289*I)], [
        (0.0736668919074069 - 0.3721976723848182*I), (0.7232010036495048
        + 0.2545999914748455*I)], [ (0.0736668919074069 +
        0.3721976723848182*I), (0.7232010036495049 - 0.2545999914748455*I
        )], [ -1, 0]];
    WeierPts:=[];

    for i in WeierPts_coor do     //make a lst of Wpoints from coordinates
    zz,pt1:=IsPoint(X,<[Re(i[1]),Im(i[1])],1>);
    zz,pt2:=IsPoint(X,<[Re(i[1]),Im(i[1])],2>);
    zz,pt3:=IsPoint(X,<[Re(i[1]),Im(i[1])],3>);
    if (Abs(Coordinates(pt1)[2]-i[2]) le 0.0001) then embed:=1;
    elif (Abs(Coordinates(pt2)[2]-i[2]) le 0.0001) then embed:=2;
    elif (Abs(Coordinates(pt3)[2]-i[2]) le 0.0001) then embed:=3;
    end if;
    zz, wpt:=IsPoint(X,<[Re(i[1]),Im(i[1])],embed>);
    WeierPts:=Append(WeierPts,wpt);     //the list of W-points
    end for;

    Fz:=function(P,n,dire)     //Fz with n the 'distance' scale and dire
        the relative direction

    for i in [1,2,3] do
    rootQ:=root(Re(Coordinates(P)[1]),Im(Coordinates(P)[1]),i);
    if (Abs(rootQ-Coordinates(P)[2]) le 0.01) then     //since Q is close
        to P, their y-coord are close.
    rinde:=i;     //the index of the y-coordinate for Q
    end if;
    end for;

    P_x_r:=Real(Coordinates(P)[1]);
    P_x_i:=Imaginary(Coordinates(P)[1]);
    Q_x_r:=P_x_r+dire[1]*10^(-n);     //real part of the x coordinate of Q
    Q_x_i:=P_x_i+dire[2]*10^(-n);
```

```
einde:=embedding(Q_x_r,Q_x_i,rinde);
zz,Q:=IsPoint(X,<[Q_x_r,Q_x_i],einde>);
gPQ:=Divisor([P,Q],[3,-1]);
distPQ:=Abs(10^(-n)*(dire[1]+dire[2]*I));

result:=normalizedtheta(gPQ)/(distPQ^3);
return result;
end function;

TX:=function(P,n,dire) //T(X)
brick1:=1;
for ele in WeierPts do     //theta part
WDIV:=Divisor([P,ele],[3,-1]);
brick1:=brick1*Abs(normalizedtheta(WDIV)^(2/27));
end for;
brick2:=Fz(P,n,dire)^(-4);
brick3:=wronskisquare(Coordinates(P)[1],Coordinates(P)[2]);
result:=brick1*brick2*brick3;
return result;
end function;

avWe:=[0.1,0.1];      //avoid Weierstrass point
dire:=[-1,0];      //initializing direction
deltapq:=6;      //initializing the 'distance' scale between P and Q
zz,PointP:=IsPoint(X,<[1+avWe[1],0+avWe[2]],1>);      //initializing
    the point P
TX(PointP,deltapq,dire)
```

IX Code for $H(X)$

```
C<I> := ComplexField(30);
Cxy<x, y> := PolynomialRing(RationalField(), 2);
f := -x^3*y+x^2*y^2-x*y^2+y^3+x^2+x;      //defining polynomial
X := RiemannSurface(f : Precision:=30);     //riemann surface
tau:=SmallPeriodMatrix(X);
Imtau:=MatrixIm(tau);
detimtau:=Determinant(Imtau);

//the following rows are rows of the big period matrix
tau1:=[-0.075648772682765897562380726 6149
    ,0.828507935186700709139824389302,
0.486565855891405863290312449179 ,-
    0.075319444360448146142631025 9273,
0.532385049449013502234878708914 , 0.231433843472050162680353039649];
tau2:=[0.486565855891405863290312449179, -
    0.075319444360448146142631025 9273,
0.500584921111822623585859133424, + 0.488740800373239742539460301357,
0.137880100924435244693062999773, -
    0.057037300117585140066233135 2196];
tau3:=[0.532385049449013502234878708914, +
    0.231433843472050162680353039649,
0.137880100924435244693062999773, -
    0.057037300117585140066233135 2196,
0.689717878372433609278715552080, +
    0.692647743504703135897856925333];

MatrixIm := function(M)     //take the imaginary part of a matrix
N := ZeroMatrix(C, Nrows(M), Ncols(M));
for i in [1..Nrows(M)] do
for j in [1..Ncols(M)] do
N[i,j] := Imaginary(M[i,j]);
end for;
end for;
return N;
end function;

rectan:=Matrix([[1.0,0.0,0.0,0.0,0.0,0.0],
    [0.0,0.0,1.0,0.0,0.0,0.0],
[0.0,0.0,0.0,0.0,1.0,0.0],
tau1,tau2,tau3]);
volume:=Abs(Determinant(rectan));

TorusTheta := function(V);     //the function ||theta|| on a torus
char:=Matrix([[0],[0],[0],[0],[0],[0]]);
```

```
37      myabstheta:=Abs(Theta(char, V, tau));
        myexponent:=-3.141592653589*Transpose(MatrixIm(V))*(Imtau^-1)*
            MatrixIm(V);
39      return Abs(Exp(myexponent[1,1])*myabstheta*detimtau^(1/4));
        end function;
41
        A:=Matrix(C,3,1,[0,0,1]);
43      B:=Matrix(C,3,1,[0,1,0]);
        C:=Matrix(C,3,1,[1,0,0]);
45      D:=Transpose(Matrix(tau[1]));
        E:=Transpose(Matrix(tau[2]));
47      F:=Transpose(Matrix(tau[3]));

49      c:=19;      \\the Riemann sum
        result:=0;
51      for ii in [1..2] do
        for jj in [1..c] do
53      for kk in [1..c] do
        for ll in [1..c] do
55      for mm in [1..c] do
        for nn in [1..c] do
57      vector:=(ii/c)*A+(jj/c)*B+(kk/c)*C+(ll/c)*D+(mm/c)*E+(nn/c)*F;
        result:=result+Log(TorusTheta(vector))/(c^6);
59      end for;
        end for;
61      end for;
        end for;
63      end for;
        end for;
65      result;
```

The code above computes the Riemann sum for $2 \times c^5$ small polyhedrons. See the end of Appendix VII for further explanation.

X Code for Klein's formula

```
C<I> := ComplexField(50);

P<U,V,W>:= PolynomialRing(Rationals(), 3);
F:= -U^3*V+U^2*V^2-U*V^2*W+V^3*W+U^2*W^2+U*W^3;
Qxy<x, y>:= PolynomialRing(Rationals(), 2);
f:= -x^3*y+x^2*y^2-x*y^2+y^3+x^2+x;     //defining polynomial
X := RiemannSurface(f);     //create the riemann surface
tau1:=SmallPeriodMatrix(X);//the period matrix

Omega1:=Matrix([[3.27458588878738559863054319936E-40 +
    1.40623766693192062162024765467*I,
-1.79952501232403458891122141246 + 1.38907786473238785380622191148*I,
-2.53736487575268201398849418408 + 0.42295457162454221877328008922 8*I
    ],[-3.32482770801607988312171989709E-40 +
    2.21299876803863343807211504318*I,
0.52029443298939017563411318001 4 + 0.31197166236570077526425813346 4*I
    ,
2.49655614620140646500159641736 - 1.72853703668803372011916427569*I
    ],[-6.11781900650324543346854241722E-40 -
    0.51924888890894515241999854511 8*I,
-0.77800185745662982300291291751 2 + 1.46695721833231037837976066119*I
    ,
3.20106058440048591056866925446 - 1.29453940681916549786848960725*I
    ]]);

//the following set S contains all the even characteristics
S:=[Matrix(RationalField(),1,[0,0,0,0,0,0,0]),Matrix(RationalField()
    ,1,[0,0,0,0,0,0,1]),Matrix(RationalField(),1,[0,0,0,0,0,1,0]),Matrix(
    RationalField(),1,[0,0,0,0,0,1,1]),Matrix(RationalField()
    ,1,[0,0,0,0,1,0,0]),Matrix(RationalField(),1,[0,0,0,0,1,0,1]),Matrix(
    RationalField(),1,[0,0,0,0,1,1,0]),Matrix(RationalField()
    ,1,[0,0,0,0,1,1,1]),Matrix(RationalField(),1,[0,0,1,0,0,0,0]),Matrix(
    RationalField(),1,[0,0,1,0,0,1,0]),Matrix(RationalField()
    ,1,[0,0,1,1,0,0,0]),Matrix(RationalField(),1,[0,0,1,1,1,0,0]),Matrix(
    RationalField(),1,[0,1,0,0,0,0,0]),Matrix(RationalField()
    ,1,[0,1,0,0,0,0,1]),Matrix(RationalField(),1,[0,1,0,1,0,0,0]),Matrix(
    RationalField(),1,[0,1,0,1,0,0,1]),Matrix(RationalField()
    ,1,[0,1,1,0,0,0,0]),Matrix(RationalField(),1,[0,1,1,1,0,0,0]),Matrix(
    RationalField(),1,[0,1,1,1,1,1,1]),Matrix(RationalField()
    ,1,[0,1,1,0,1,1,1]),Matrix(RationalField(),1,[1,0,0,0,0,0,0]),Matrix(
    RationalField(),1,[1,0,0,0,0,0,1]),Matrix(RationalField()
    ,1,[1,0,0,0,0,1,0]),Matrix(RationalField(),1,[1,0,0,0,0,1,1]),Matrix(
    RationalField(),1,[1,0,1,0,0,0,0]),Matrix(RationalField()
    ,1,[1,0,1,0,1,0,0]),Matrix(RationalField(),1,[1,0,1,1,0,0,1]),Matrix(
```

```
        RationalField(),1,[1,0,1,1,1,1]),Matrix(RationalField()
        ,1,[1,1,0,0,0,0]),Matrix(RationalField(),1,[1,1,0,0,0,1]),Matrix(
        RationalField(),1,[1,1,0,1,1,0]),Matrix(RationalField()
        ,1,[1,1,0,1,1,1]),Matrix(RationalField(),1,[1,1,1,0,0,0]),Matrix(
        RationalField(),1,[1,1,1,0,1,1]),Matrix(RationalField()
        ,1,[1,1,1,1,1,0]),Matrix(RationalField(),1,[1,1,1,1,0,1])];

MatrixIm := function(M) // Imaginary part of a matrix
N := ZeroMatrix(C, Nrows(M), Ncols(M));
for i in [1..Nrows(M)] do
for j in [1..Ncols(M)] do
N[i,j] := Imaginary(M[i,j]);
end for;
end for;
return N;
end function;

Klein_ratio:=function(period)//compute the ratio of Klein formula
chitilde:=1;
zerovector:=Matrix(C,1,[0,0,0]);
for ele in S do
chitilde:=chitilde*Theta((1/2)*ele,zerovector,period);
end for;
left:=DiscriminantOfTernaryQuartic(F)^2;
right:=chitilde*(2*3.1415926)^(54)/(2^(28)*Determinant(Omega1)^(18));
return right/left;
end function;

Klein_ratio(tau1);
```