



Universiteit
Leiden
The Netherlands

Explicit computation of the height of a Gross-Schoen Cycle

Wang, R.

Citation

Wang, R. (2022, October 18). *Explicit computation of the height of a Gross-Schoen Cycle*. Retrieved from <https://hdl.handle.net/1887/3480346>

Version: Publisher's Version

[Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

License: <https://hdl.handle.net/1887/3480346>

Note: To cite this publication please use the final published version (if applicable).

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

```
1:  $r_3 = -\frac{1}{2} + \frac{\sqrt{3}}{2}i$ 
2:  $u = \frac{9abc - 27a^2d - 2b^3}{54a^3}$ 
3:  $v = \frac{(12ac^3 - 3b^2c^2 - 54abcd + 81a^2d^2 + 12b^3d)^{1/2}}{18a^2}$ 
4: if  $|u + v| \geq |u - v|$  then
5:    $m = (u + v)^{(1/3)}$ 
6: else
7:    $m = (u - v)^{(1/3)}$ 
8: end if
9: if  $m = 0$  then
10:   $n = 0$ 
11: else
12:   $n = \frac{b^2 - 3ac}{9am}$ 
13: end if
14: return  $r_3^{i-1}m + r_3^{2i-2}n - \frac{b}{3a}$ 
```

APPENDIX

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:

```

1   a:=1;
2   b:=x^2-x;
3   c:=-x^3;
4   d:=x^2+x;
5   cubic_unit:=-1/2+3^(1/2)/2*I;
6   root:=function(u,v,i);
7   numa:=a;
8   numb:=Evaluate(b,[u+v*I,0]);
9   numc:=Evaluate(c,[u+v*I,0]);
10  preuu:=(9*a*b*c-27*a^2*d-2*b^3)/(54*a^3);
11  uu:=Evaluate(preuu,[u+v*I,0]);
12  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+
13      v*I,0]);
14  vv:=(prevv^(1/2))/(18*a^3);
15  m:=(uu-vv)^(1/3);
16  if Abs(uu-vv) ge Abs(uu-vv) then
17      m:=(uu+v*v)^(1/3);
18  end if;
19  n:=0;
20  if m ne 0 then
21      n:=(numb^2-3*numa*numc)/(9*numa*m);
22  end if;
23  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:

$$\begin{aligned} \mathfrak{W} = \{ & \\ & (0.0000000000, 0.0000000000), \\ & (-2.104587156, -0.3159811808), \\ & (-1.241974125, -1.824965133), \\ & (0.3539677313, -0.7304015296), \\ & (-1.959274470, -1.410535295), \\ & (-1.0000000000, 0.0000000000), \\ & (-1.455757161 - 1.277524807 * I, -0.5258271579 - 3.390009274 * I), \\ & (-1.455757161 + 1.277524807 * I, -0.5258271579 + 3.390009274 * I), \\ & (1.266455202 - 1.314861880 * I, 0.1352071928 + 2.966892635 * I), \\ & (1.266455202 + 1.314861880 * I, 0.1352071928 - 2.966892635 * I), \\ & (0.2089717154 - 1.597843531 * I, 0.1450907026 - 2.034168656 * I), \\ & (0.2089717154 + 1.597843531 * I, 0.1450907026 + 2.034168656 * I), \\ & (-1.319681529 - 0.4985373090 * I, -0.4698939284 - 0.6348619710 * I), \\ & (-1.319681529 + 0.4985373090 * I, -0.4698939284 + 0.6348619710 * I), \\ & (1.691125343 - 0.6842628628 * I, 0.6312556990 + 0.6046227212 * I), \\ & (1.691125343 + 0.6842628628 * I, 0.6312556990 - 0.6046227212 * I), \\ & (-1.125631934 - 0.2207052168 * I, -0.6596966755 + 0.07228789093 * I), \\ & (-1.125631934 + 0.2207052168 * I, -0.6596966755 - 0.07228789093 * I), \\ & (-0.5282412488 - 1.005235134 * I, -0.4177679369 + 0.3210649478 * I), \\ & (-0.5282412488 + 1.005235134 * I, -0.4177679369 - 0.3210649478 * I), \\ & (-0.3349732907 - 0.4139797966 * I, -0.4206274002 - 0.9204898342 * I), \\ & (-0.3349732907 + 0.4139797966 * I, -0.4206274002 + 0.9204898342 * I), \\ & (0.07366689191 - 0.3721976724 * I, 0.7232010036 + 0.2545999915 * I), \\ & (0.07366689191 + 0.3721976724 * I, 0.7232010036 - 0.2545999915 * I) \end{aligned}$$

III Code for canonical form

```

1   from sage.schemes.riemann_surfaces.riemann_surface import
2       RiemannSurface
3   R.<x,y> = QQ[]
4   f = -x^3*y+x^2*y^2-x*y^2+y^3+x^2+x
5
6   S = RiemannSurface(f)
7   M = S.period_matrix()
8   Omega1 = M[[0,1,2],[0,1,2]]
9   Omega2 = M[[0,1,2],[3,4,5]]
10  A = Omega1.apply_map(real)
11  B = Omega1.apply_map(imag)
12  tau = S.riemann_matrix()
13  Omega1bar= A-B*I
14  Imtau = tau.apply_map(imag)
15  RieSecBil = Omega1bar*Imtau*Omega1.transpose()
16
17  [ 8.14058999836226 - 4.44089209850063e-16*I -2.71111012021317 -
18    9.21762666195036e-14*I -5.21995559097122 - 5.85642645489770e-14*I
19    ]
20  [-2.71111012021315 + 6.52811138479592e-14*I 8.66278158093292 +
21    8.32667268468867e-14*I 5.65493670273040 + 1.56541446472147e-13*I
22    ]
23  [-5.21995559097118 + 1.45994327738208e-14*I 5.65493670273037 -
24    1.75831571525009e-14*I 10.7557438794561 + 1.18932641512970e-14*I
25    ]
26
27  w11 = RieSecBil[0][0]
28  w1y = RieSecBil[0][1]
29  w1x = RieSecBil[0][2]
30  wyy = RieSecBil[1][1]
31  wyx = RieSecBil[1][2]
32  wxx = RieSecBil[2][2]
33  k = (wyx-w1x*w1y/w11)/(wyy-w1y*w1y/w11)
34  norm1 = RieSecBil[0][0]**(1/2)
35  norm2 = (wyy-(2*w1y*w1y/w11)+w1y*w1y/w11)**(1/2)
36  norm3 = (wxx+w1x*w1x/w11+k*k*wyy+k*k*w1y*w1y/w11-2*w1x*w1x/w11-2*k*
37    wyx+2*k*w1y*w1x/w11+2*k*w1x*w1y/w11-2*k*w1x*w1y/w11-2*k*k*w1y*w1y
38    /w11)**(1/2)

```

Gram-Schmidt process

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

IV Code for semistability

```

1   from collections import defaultdict
2   PP.<x,y,z> = ProjectiveSpace(QQ,2)
3   Poly.<U,V> = PolynomialRing(ZZ, 2)
4   RR.<X, Y, Z> = PolynomialRing(ZZ, 3)
5   C = Curve(-x^3*y+x^2*y^2-x*y^2*z+y^3*z+x^2*z^2+x*z^3, PP)
6   inj1 = Poly.hom([X, Y], RR)
7
8
9   def IsDegree1(I):      #checking the ideal I has degree 1 generators
10  out = True
11  for f in I.gens():
12    if not f.degree() == 1:
13      out = False
14      return out
15
16
17  def MyBadPrimes(C):      #finding out primes with bad reduction
18  f = C.defining_polynomial()
19  RZ.<xZ,yZ,zZ> = PolynomialRing(ZZ, 3)
20  coeffs = f.coefficients()
21  dens = [c.denominator() for c in coeffs]
22  den = lcm(dens)
23  F = RZ(f*den)
24  Fx = F.derivative(xZ)
25  Fy = F.derivative(yZ)
26  Fz = F.derivative(zZ)
27  NaiveDisc = 1
28  for P in [[xZ,yZ,1],[xZ,1,zZ],[1,yZ,zZ]]:
29    I = ideal([g(P) for g in [F,Fx,Fy,Fz]])
30    G = I.groebner_basis()
31    n = G[len(G)-1]
32    NaiveDisc = lcm(n, NaiveDisc)
33  return [a[0] for a in factor(NaiveDisc)]
34
35
36  def MyBadPoints(C, default_prime):      #CORE FUNCTION: finding out
37    all singular points over certain prime number and the tangent
38    lines
39  f = C.defining_polynomial()
40  RZ.<xZ,yZ,zZ> = PolynomialRing(ZZ, 3)
41  coeffs = f.coefficients()
42  dens = [c.denominator() for c in coeffs]
43  den = lcm(dens)
44  F = RZ(f*den)

```

APPENDIX

```
43 Fx = F.derivative(xZ)
44 Fy = F.derivative(yZ)
45 Fz = F.derivative(zZ)
46 Out = [ 'Dimention of singular locus is positive.']
47 GG = [] #list of associate primes (after field
48 extension)
49 sing_set = [] #list of degree 2 parts of singular points
50 after transition
51 factor_sing = [] #list of factorizations of degree 2 parts
52 degree = 1
53 SingDim = 0
54 found = False
55 while found == False:
56     Rp = RZ.change_ring(GF(default_prime^degree))
57     win = True
58     S.<u, v> = PolynomialRing(GF(default_prime^degree), 2)
59     for P in [ [u, v, 1], [u, 1, v], [1, u, v] ]:
60         affine_patch_map = RZ.hom(P, S)
61         I_2 = Ideal(S,[ affine_patch_map(g) for g in [F, Fx, Fy, Fz]])
62         G_2 = I_2.associated_primes()
63         for I in G_2:
64             VV = I.gens()
65             if Set(VV).cardinality() == 1:
66                 SingDim = 1
67             if SingDim == 1:
68                 break
69             for I in G_2:
70                 if IsDegree1(I) == False:
71                     win = False #This means that the singular points don't have
72                     coordinates in this base field and we still need to extend it
73                     if SingDim == 1:
74                         break
75                     Out = []
76                     if win == True:
77                         for P in [ [u, v, 1], [u, 1, v], [1, u, v] ]:
78                             affine_patch_map = RZ.hom(P, S)
79                             I_2 = Ideal(S,[affine_patch_map(g) for g in [F, Fx, Fy, Fz]])
80                             G_2 = I_2.associated_primes()
81                             GG += G_2
82                             for AP in G_2: #in this for loop, we find out the coordinate
83                                 of singular points
84                             T.<x, y> = S.quotient(ideal(S, [w for w in AP.gens()]))
85                             Quot = T.cover()
86                             if Quot(1) == 0:
87                                 continue
88                             for m in GF(default_prime^degree):
89                                 if Quot(u-m) == 0:
```

```

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

```

V Code for semi-canonical divisor

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

```

1 PeriodMatrix:=Matrix([
2   [-0.0756487726827658975 + 0.82850793518670070913*I,
3     0.48656585589140586329 - 0.07531944436044814614*I,
4     0.53238504944901350223 + 0.23143384347205016268*I],
5   [0.48656585589140586329 - 0.07531944436044814614*I,
6     0.50058492111822623585 + 0.48874080037323974253*I,
7     0.137880100924435244693 - 0.05703730011758514006*I],
8   [0.53238504944901350223 + 0.23143384347205016268*I,
9     0.137880100924435244693 - 0.05703730011758514006*I,
10    0.689717878372433609278 + 0.69264774350470313589*I]])

```

```

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

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

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

16 MatrixIm := function(M) //function to take the imaginary part of a
17   matrix.
18 N := ZeroMatrix(C, Nrows(M), Ncols(M));
19   for i in [1..Nrows(M)] do
20     for j in [1..Ncols(M)] do
21       N[i,j] := Imaginary(M[i,j]);
22     end for;
23   end for;
24   return N;
25 end function;

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

```

```

28      out := [];
29      for i in [1..Ncols(M)] do
30          out := out cat [ColumnSubmatrix(M, i, 1)];
31      end for;
32      return out;
33      end function;

34
35      char:=Matrix([[0],[0],[0],[0],[0],[0]]);
36      Imtau:=MatrixIm(tau);
37      detimtau:=Determinant(Imtau);
38      LeftPeriod:=RemoveColumn(RemoveColumn(RemoveColumn(BigPeriodMatrix(X)
39          ,4),4));
40      ID3 := DiagonalMatrix(C, [1,1,1]);
41      columnlist := Columns(ID3) cat Columns(tau);
42      AJCD := Matrix(AbelJacobi(CanDiv)); //the image of canonical
43          divisor under AJ map

44      AJ_2_torsion := [];
45      for v in CartesianPower([0,1], 6) do
46          new := ZeroMatrix(C, 3, 1);
47          for i in [1..6] do
48              new := new + v[i] * columnlist[i];
49          end for;
50          AJ_2_torsion := AJ_2_torsion cat [(1/2) * new];
51      end for;

52      translates_to_try := []; //this will be a list of all 64 square
53          roots of CanDiv
54      for V in AJ_2_torsion do
55          translates_to_try := translates_to_try cat [(1/2) * AJCD + V];
56      end for;

57      torus_theta_function := function(D)// D is a divisor of degree g-1
58          myinput := Matrix(AbelJacobi(D));
59          myabsttheta:=Abs(Theta(char, myinput, tau));
60          myiminput:= MatrixIm(myinput);
61          myexponent:=-3.141592653589*Transpose(myiminput)*(Imtau^(-1))*myiminput;
62          return Exp(myexponent[1,1])*myabsttheta*detimtau^(1/4);
63      end function;

64      find_correct_theta_translate := function(D) //find out the correct
65          translation
66      P := AbelJacobi(D);
67      out := [];
68      for V in translates_to_try do
69          myinput := Matrix(P) - V;

```

APPENDIX

```
70 myabstheta:=Abs(Theta(char , myinput , tau));
71 myiminput:= MatrixIm(myinput);
72 myexponent:=-3.141592653589*Transpose(myiminput)*(Imtau^(-1))*myiminput;
73 if Abs(Exp(myexponent[1,1])*myabstheta*detimtau^(1/4)) <= 0.00000001
74     then
75         return V;
76     end if;
77 end for;
78 end function;
79
80 //we check the result by choosing 3 different effective divisors
81 ThetaTranslate1 := find_correct_theta_translate(Div);
82 ThetaTranslate1;
83 ThetaTranslate2 := find_correct_theta_translate(Div2);
84 ThetaTranslate2;
85 ThetaTranslate3 := find_correct_theta_translate(Div3);
86 ThetaTranslate3;
```

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

```

1 C<I> := ComplexField(50);
2 Qxy<x, y> := PolynomialRing(Rationals(), 2);
3 f := -x^3*y+x^2*y^2-x*y^3+y^3+x^2+y; //defining polynomial
4 X := RiemannSurface(f); //create the riemann surface
5
6 tau1:=SmallPeriodMatrix(X); //period matrix implemented in Magma
7 tau2:=Matrix(C,3,[ 0.855638485763810 + 1.03033263075936*I,
8   0.380495294693576 + 0.00436338826915927*I, 0.506949568788359 -
9   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
11
12 //the following set S contains all the even characteristics
13 S:=[Matrix(RationalField(),1,[0,0,0,0,0,0]),Matrix(RationalField()
14   ,1,[0,0,0,0,0,1]),Matrix(RationalField(),1,[0,0,0,0,1,0]),Matrix(
15   RationalField(),1,[0,0,0,0,1,1]),Matrix(RationalField()
16   ,1,[0,0,0,1,0,0]),Matrix(RationalField(),1,[0,0,0,1,0,1]),Matrix(
17   RationalField(),1,[0,0,0,1,1,0]),Matrix(RationalField()
18   ,1,[0,0,0,1,1,1]),Matrix(RationalField(),1,[0,0,1,0,0,0]),Matrix(
19   RationalField(),1,[0,0,0,1,0,1]),Matrix(RationalField()
20   ,1,[0,0,1,0,1,0]),Matrix(RationalField(),1,[0,0,1,1,0,0]),Matrix(
21   RationalField(),1,[0,1,0,0,0,0]),Matrix(RationalField()
22   ,1,[0,1,0,0,0,1]),Matrix(RationalField(),1,[0,1,0,1,0,0]),Matrix(
23   RationalField(),1,[0,1,0,1,0,1]),Matrix(RationalField()
24   ,1,[0,1,1,0,0,0]),Matrix(RationalField(),1,[0,1,1,1,0,0]),Matrix(
25   RationalField(),1,[0,1,1,1,1,0]),Matrix(RationalField()
26   ,1,[0,1,1,0,1,1]),Matrix(RationalField(),1,[1,0,0,0,0,0]),Matrix(
27   RationalField(),1,[1,0,0,0,0,1]),Matrix(RationalField()
28   ,1,[1,0,0,0,1,0]),Matrix(RationalField(),1,[1,0,0,0,1,1]),Matrix(
29   RationalField(),1,[1,0,1,0,0,0]),Matrix(RationalField()
30   ,1,[1,0,1,0,0,1]),Matrix(RationalField(),1,[1,0,1,1,0,0]),Matrix(
31   RationalField(),1,[1,0,1,1,0,1]),Matrix(RationalField()
32   ,1,[1,0,1,1,1,0]),Matrix(RationalField(),1,[1,0,1,1,1,1]),Matrix(
33   RationalField(),1,[1,1,0,0,0,0]),Matrix(RationalField()
34   ,1,[1,1,0,0,0,1]),Matrix(RationalField(),1,[1,1,0,1,0,0]),Matrix(
35   RationalField(),1,[1,1,0,1,0,1]),Matrix(RationalField()
36   ,1,[1,1,0,1,1,0]),Matrix(RationalField(),1,[1,1,1,0,0,0]),Matrix(
37   RationalField(),1,[1,1,1,0,0,1]),Matrix(RationalField()
38   ,1,[1,1,1,0,1,0]),Matrix(RationalField(),1,[1,1,1,1,0,0]),Matrix(
39   RationalField(),1,[1,1,1,1,0,1])];
40
41 MatrixIm := function(M) //imaginary part of a matrix
42 N := ZeroMatrix(C, Nrows(M), Ncols(M));
43 for i in [1..Nrows(M)] do

```

APPENDIX

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

25      compute:=function(period) //compute || theta ||
26          chitilde:=1;
27          zerovector:=Matrix(C,1,[0,0,0]);
28          for ele in S do
29              chitilde:=chitilde*Theta((1/2)*ele,zerovector,period);
30          end for;
31          abschitilde:=Abs(chitilde);
32          Imtau:=MatrixIm(period);
33          det:=Determinant(Imtau);
34          prereresult:=Log((det^9)*abschitilde);
35          prereresult:=26*Log(2)+54*Log(Pi(C))+prereresult;
36          result:=-(21/18)*prereresult; //contribution of \chi_18 at the
37          infinite place
38          return result;
39      end function;

40      //we compute the result using two period matrices implemented in
41      //Magma and SageMath
42      compute(tau1);
43      compute(tau2);
```

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

```

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

8    MatrixIm := function(M) //take the imaginary part of a matrix
9    N := ZeroMatrix(C, Nrows(M), Ncols(M));
10   for i in [1..Nrows(M)] do
11     for j in [1..Ncols(M)] do
12       N[i, j] := Imaginary(M[i, j]);
13     end for;
14   end for;
15   return N;
16 end function;

18   Imtau:=MatrixIm(tau);
19   detimtau:=Determinant(Imtau);
20   ThetaTranslate:=Matrix([0.479250542651680186758281300774 -
21                         0.00334176833187451614116524746349*I],
22                         [0.698684877508432322290935180841 + 0.199495723882563563098727841525*
23                           I],
24                         [0.00722266620787249384535143960054 -
25                           0.0430102069343208149623250436805*I]);
26

28   normalizedtheta := function(D); //compute || theta ||_{g-1} at D
29   P := AbelJacobi(D);
30   myinput := Matrix(P) - ThetaTranslate;
31   char:=Matrix([[0],[0],[0],[0],[0],[0]]);
32   myabsttheta:=Abs(Theta(char, myinput, tau));
33   myiminput:= MatrixIm(myinput);
34   myexponent:=-3.141592653589*Transpose(myiminput)*(Imtau^-1)*myiminput
35   ;
36   return Abs(Exp(myexponent[1,1])*myabsttheta*detimtau^(1/4));
37 end function;

39   a:=1;
40   b:=x^2-x;
41   c:=-x^3;
42   d:=x^2+x;
43   cubic_unit:=-1/2+3^(1/2)/2*I;

```

APPENDIX

```

40      root:=function(u,v,i);      //solve the cubic equation with
        coefficients a, b, c and d.
41      numa:=a;
42      numb:=Evaluate(b,[u+v*I,0]);
43      numc:=Evaluate(c,[u+v*I,0]);
44      preuu:=(9*a*b*c-27*a^2*d-2*b^3)/(54*a^3);
45      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)),[
47          u+v*I,0]);
48      vv:=(prevv^(1/2))/(18*a^2);
49      m:=(uu-vv)^(1/3);
50      if Abs(uu+vv) ge Abs(uu-vv) then
51          m:=(uu+vv)^(1/3);
52      end if;
53      n:=0;
54      if (Abs(m) ge 0.00001) then
55          n:=(numb^2-3*numa*numc)/(9*numa*m);
56      else n:=0;
57      end if;
58      result:=(cubic_unit^(i-1))*m+n*cubic_unit^(2*i-2)-(numb)/(3*numa);
59      return result;
60  end function;

61
62      embedding:=function(u,v,i);      //find out the embedding index in
        Magma
63      zz,pt1:=IsPoint(X,<[u,v],1>);
64      zz,pt2:=IsPoint(X,<[u,v],2>);
65      zz,pt3:=IsPoint(X,<[u,v],3>);
66      if (Abs(Coordinates(pt1)[2]-root(u,v,i)) le 0.0001) then result:=1;
67      elif (Abs(Coordinates(pt2)[2]-root(u,v,i)) le 0.0001) then result:=2;
68      elif (Abs(Coordinates(pt3)[2]-root(u,v,i)) le 0.0001) then result:=3;
69      end if;
70      return result;
71  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)
73      zz, pt_in_cover_i:=IsPoint(X,<[u,v],embedding(u,v,i)>);
74      targetdivisor:=Divisor([ BP,pt_in_cover_i],[3,-1]);
75      return normalizedtheta(targetdivisor);
76  end function;

77
78      omegal := function(u,v,i)      // first differential form, i denotes
        index of y
79      u:=u+0.000001;
80      v:=v+0.000001;
81      x:=u+v*I;

```

```

82      y:=root(u,v,i);
83      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
87          index of y
88          u:=u+0.000001;
89          v:=v+0.000001;
90          x:=u+v*I;
91          y:=root(u,v,i);
92          return x/(-x^3+2*x^2*y-2*x*y+3*y^2);
93          end function;

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

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

113     ff:=function(u,v)      //compute three i in one time
114         result:= Log(theta_wrt_yi(u,v,1))*(canonicalform(u,v,1))+Log(
115             theta_wrt_yi(u,v,2))*(canonicalform(u,v,2))+Log(theta_wrt_yi(u,v
116             ,3))*(canonicalform(u,v,3));
117         return result;
118         end function;

119     SX:=0;      //take Riemann sum
120     scale:=0.01;
121     rex_start:=-10;
122     imx_start:=-10;
123     for p in [1..200] do
124         for q in [1..2000] do

```

```
122     SX:=SX+ff (p*scale+rex_start+0.005,q*scale+imx_start+0.005)*scale*  
           scale*(-9);  
    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 $T(X)$

```

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

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

17 Imtau:=MatrixIm(tau);
18 detimtau:=Determinant(Imtau);
19 ThetaTranslate:=Matrix([[0.479250542651680186758281300774 -
20   0.00334176833187451614116524746349*I],
21 [0.698684877508432322290935180841 + 0.199495723882563563098727841525*
22   I],
23 [0.00722266620787249384535143960054 -
24   0.0430102069343208149623250436805*I]]);
25
26 normalizedtheta := function(D);      // || theta ||_{g-1}
27 P := AbelJacobi(D);
28 myinput := Matrix(P) - ThetaTranslate;
29 char:=Matrix([[0],[0],[0],[0],[0],[0]]);
30 myabsttheta:=Abs(Theta(char, myinput, tau));
31 myiminput:= MatrixIm(myinput);
32 myexponent:=-3.141592653589*Transpose(myiminput)*(Imtau^-1)*myiminput
33   ;
34   return Abs(Exp(myexponent[1,1])*myabsttheta*detimtau^(1/4));
35 end function;

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

```

APPENDIX

```

39      root:=function(u,v,i);      // solve equation with coefficients a,b,c,d
40      numa:=a;
41      numb:=Evaluate(b,[u+v*I,0]);
42      numc:=Evaluate(c,[u+v*I,0]);
43      preuu:=(9*a*b*c-27*a^2*d-2*b^3)/(54*a^3);
44      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)),[
46          u+v*I,0]);
47      vv:=(prevv^(1/2))/(18*a^2);
48      m:=(uu-vv)^(1/3);
49      if Abs(uu+vv) ge Abs(uu-vv) then
50          m:=(uu+vv)^(1/3);
51      end if;
52      n:=0;
53      if m ne 0 then
54          n:=(numb^2-3*numa*numc)/(9*numa*m);
55      end if;
56      result:=(cubic_unit^(i-1))*m+n*cubic_unit^(2*i-2)-(numb)/(3*numa);
57      return result;
58  end function;

59  wronskisquare:=function(x,y)
60      fy:=-x^3+2*x^2*y-2*x*y+3*y^2;      // partial derivative of defining
61      polynomial wrt y
62      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);      //
63      implicit derivative of y wrt x
64      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*
65          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);      //
66      second derivative
67      dfy:=-3*x^2+4*x*y+2*x^2*dy-2*y-2*x*dy+6*y*dy;      // implicit
68      derivative of fy wrt x
69      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*
70          ddy;      // second derivative
71      wl:=1/fy;      // differential forms
72      wy:=y/fy;
73      wx:=x/fy;
74      dwl:=-dfy/fy^2;      // implicit derivative of wl wrt x
75      dwy:=(dy*fy-dfy*y)/(fy^2);
76      dwx:=(fy-x*dfy)/(fy^2);
77      ddwl:=-(ddfy*fy^2-2*fy*dfy^2)/(fy^4);      // second derivative
78      ddwy:=((ddy*fy+dy*dfy-ddfy*y-dfy*dy)*fy^2-2*fy*dfy*(dy*fy-dfy*y))/(fy
79          ^4);
80      ddwx:=((dfy-dfy-x*ddfy)*fy^2-2*fy*dfy*(fy-x*dfy))/(fy^4);
81      c11:=0.350487116953118;      // orthogonalizing matrix
82      c21:=0.119553875346235;
83      c22:=0.358981759779085;

```

```

77      c31:=0.203008239643111;
79      c32:=-0.216555180015011;
81      c33:=0.429067210690657;
83      W11:=c11*w1; //following lines gives the wronskian matrix wrt a
85      orthonormal basis of differential form.
87      W12:=c21*w1+c22*wy;
89      W13:=c31*w1+c32*wy+c33*wx;
91      W21:=c11*dw1;
93      W22:=c21*dw1+c22*dwy;
95      W23:=c31*dw1+c32*dwy+c33*dwx;
97      W31:=c11*ddw1;
99      W32:=c21*ddw1+c22*ddwy;
101     W33:=c31*ddw1+c32*ddwy+c33*ddwx;
103    Wronski:=Matrix ([[W11,W12,W13],[W21,W22,W23],[W31/2,W32/2,W33/2]]);
105    detwro:=Abs(Determinant(Wronski));
107    return detwro^2;
109    end function;

embedding:=function(u,v,i); //this gives the embedding index of
111   point (u+v*I,y_i) on \cC implemented in Magma
113   zz,pt1:=IsPoint(X,<[u,v],1>);
115   zz,pt2:=IsPoint(X,<[u,v],2>);
117   zz,pt3:=IsPoint(X,<[u,v],3>);
119   if (Abs(Coordinates(pt1)[2]-root(u,v,i)) le 0.01) then result:=1;
121   elif (Abs(Coordinates(pt2)[2]-root(u,v,i)) le 0.01) then result:=2;
123   elif (Abs(Coordinates(pt3)[2]-root(u,v,i)) le 0.01) then result:=3;
125   end if;
127   return result;
129   end function;

131 WeierPts_coor:=[[ 0, 0], [ -2.104587155963303, -0.3159811807558051],
133   [ (-1.455757161115023 - 1.277524806650578*I),
135   (-0.5258271579143416 - 3.390009274518869*I)], [ -
137   (-1.455757161115023 + 1.277524806650578*I), (-0.5258271579143403
139   + 3.390009274518867*I)], [ -1.95927446954141,
141   -1.410535295372148], [ (1.266455202041324 - 1.31486188027894*I),
143   (0.1352071928093522 + 2.966892634578606*I)], [ (1.266455202041324
145   + 1.31486188027894*I), (0.1352071928093521 - 2.966892634578606*I
147)], [ (0.2089717154013823 - 1.59784353074737*I),
149   (0.1450907026008794 - 2.034168656250672*I)], [ -
151   (0.2089717154013823 + 1.59784353074737*I), (0.1450907026008794 +
153   2.034168656250671*I)], [ (-1.31968152920268 - 0.4985373089678906*I),
155   (-0.4698939283630282 - 0.634861971042462*I)], [ -
157   (-1.31968152920268 + 0.4985373089678906*I), (-0.4698939283630282
159   + 0.6348619710424618*I)], [ (1.691125343336006 -
161   0.6842628628388769*I), (0.6312556990319607 + 0.6046227212351708*I
163)], [ (1.691125343336006 + 0.6842628628388769*I),

```

APPENDIX

```

(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:=[];
107
for i in WeierPts_coor do //make a lst of Wpoints from coordinates
109    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;
113    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>);
117    WeierPts:=Append(WeierPts,wpt); //the list of W-points
end for;
119
Fz:=function(P,n,dire) //Fz with n the 'distance' scale and dire
    the relative direction
121
for i in [1,2,3] do
rootQ:=root(Re(Coordinates(P)[1]),Im(Coordinates(P)[1]),i);
123    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
125    end if;
end for;
127
P_x_r:=Real(Coordinates(P)[1]);
P_x_i:=Imaginary(Coordinates(P)[1]);
129    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);
131
133

```

```

135      einde:=embedding(Q_x_r,Q_x_i,rinde);
136      zz ,Q:=IsPoint(X,<[Q_x_r,Q_x_i] ,einde>);
137      gPQ:=Divisor ([P,Q],[3,-1]);
138      distPQ:=Abs(10^(-n)*( dire [1]+ dire [2]*I));
139
140      result:=normalizedtheta (gPQ) /(distPQ ^3);
141      return result;
142      end function;
143
143 TX:=function (P,n,dire) //T(X)
144     brick1:=1;
145     for ele in WeierPts do //theta part
146       WDIV:=Divisor ([P,ele],[3,-1]);
147       brick1:=brick1*Abs(normalizedtheta (WDIV)^(2/27));
148     end for;
149     brick2:=Fz(P,n,dire)^(-4);
150     brick3:=wronskisquare(Coordinates(P) [1],Coordinates(P) [2]);
151     result:=brick1*brick2*brick3;
152     return result;
153   end function;
154
155 avWe:=[0.1,0.1]; //avoid Weierstrass point
156 dire:=[-1,0]; //initializing direction
157 deltapq:=6; //initializing the 'distance' scale between P and Q
158 zz ,PointP:=IsPoint(X,<[1+avWe[1],0+avWe[2]],1>); //initializing
159           the point P
TX(PointP,deltapq,dire)

```

IX Code for $H(X)$

```

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

9 //the following rows are rows of the big period matrix
10 tau1:=[-0.0756487726827658975623807266149
11 ,0.828507935186700709139824389302,
12 0.486565855891405863290312449179 ,-
13 0.0753194443604481461426310259273,
14 0.532385049449013502234878708914 , 0.231433843472050162680353039649];
15 tau2:=[0.486565855891405863290312449179 ,-
16 0.0753194443604481461426310259273,
17 0.500584921111822623585859133424, + 0.488740800373239742539460301357 ,
18 0.137880100924435244693062999773, -
19 0.0570373001175851400662331352196];
20 tau3:=[0.532385049449013502234878708914 , +
21 0.231433843472050162680353039649,
22 0.137880100924435244693062999773, -
23 0.0570373001175851400662331352196 ,
24 0.689717878372433609278715552080, +
25 0.692647743504703135897856925333];

27 MatrixIm := function(M) //take the imaginary part of a matrix
28 N := ZeroMatrix(C, Nrows(M), Ncols(M));
29 for i in [1..Nrows(M)] do
30 for j in [1..Ncols(M)] do
31 N[i,j] := Imaginary(M[i,j]);
32 end for;
33 end for;
34 return N;
35 end function;

37 rectan:=Matrix([[1.0,0.0,0.0,0.0,0.0,0.0],
38 [0.0,0.0,1.0,0.0,0.0,0.0],
39 [0.0,0.0,0.0,0.0,1.0,0.0],
40 tau1,tau2,tau3]);
41 volume:=Abs(Determinant(rectan));

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

```

37 myabstheta:=Abs(Theta(char , V, tau));
38 myexponent:=-3.141592653589*Transpose(MatrixIm(V))*(Imtau^-1)*
39 MatrixIm(V);
40 return Abs(Exp(myexponent [1 ,1])*myabstheta*detimtau^(1/4));
41 end function;

42
43 A:=Matrix(C,3 ,1 ,[0 ,0 ,1]);
44 B:=Matrix(C,3 ,1 ,[0 ,1 ,0]);
45 C:=Matrix(C,3 ,1 ,[1 ,0 ,0]);
46 D:=Transpose(Matrix(tau [1]));
47 E:=Transpose(Matrix(tau [2]));
48 F:=Transpose(Matrix(tau [3]));

49 c:=19;      \\the Riemann sum
50 result:=0;
51 for ii in [1..2] do
52   for jj in [1..c] do
53     for kk in [1..c] do
54       for ll in [1..c] do
55         for mm in [1..c] do
56           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;
58             result:=result+Log(TorusTheta(vector))/(c^6);
59           end for;
60         end for;
61       end for;
62     end for;
63   end for;
64 result;
65

```

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

```

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

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

9 Omega1:=Matrix([[3.27458588878738559863054319936E-40 +
11 1.40623766693192062162024765467*I,
-1.79952501232403458891122141246 + 1.38907786473238785380622191148*I,
-2.53736487575268201398849418408 + 0.422954571624542218773280089228*I
13 ],[-3.32482770801607988312171989709E-40 +
2.21299876803863343807211504318*I,
0.520294432989390175634113180014 + 0.311971662365700775264258133464*I
15 ,
2.49655614620140646500159641736 - 1.72853703668803372011916427569*I
],[-6.1178190065032453346854241722E-40 -
0.519248888908945152419998545118*I,
17 -0.778001857456629823002912917512 + 1.46695721833231037837976066119*I
,
3.20106058440048591056866925446 - 1.29453940681916549786848960725*I
]]);

//the following set S contains all the even characteristics
19 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,[0,1,1,1,1,0]),Matrix(RationalField()
,1,[0,1,1,1,1,1]),Matrix(RationalField()
,1,[1,0,0,0,0,0]),Matrix(RationalField(),1,[1,0,0,0,0,1]),Matrix(
RationalField(),1,[1,0,0,1,0,0]),Matrix(RationalField()
,1,[1,0,0,1,0,1]),Matrix(RationalField(),1,[1,0,0,1,1,0]),Matrix(
RationalField(),1,[1,0,1,0,0,0]),Matrix(RationalField(),1,[1,0,1,0,0,1]),
21 ,1,[1,0,1,0,1,0]),Matrix(RationalField(),1,[1,0,1,1,0,0]),Matrix(
RationalField(),1,[1,0,1,1,0,1]),Matrix(RationalField())
,1,[1,0,1,1,1,0]),Matrix(RationalField(),1,[1,0,1,1,1,1])];

```

```

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]) ];

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

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

41 Klein_ratio(tau1);

```

APPENDIX