Translational and minimal surfaces-v1.0

September 2, 2021

1 Sage Notebook: Affine equivalences of surfaces of translation and minimal surfaces, and applications to symmetry detection and design

This is Sage code accompanying the paper [1].

[1] Juan Gerardo Alcazar, Georg Muntingh. *Affine equivalences of rational surfaces of translation, and applications to rational minimal surfaces.* Available at https://arxiv.org/abs/2103.00151

1.1 Helper functions

We start by introducing some helper functions that provide basic operations for the vector datatype.

[1]:
```
import matplotlib.pyplot as plt

def normc2(c):
    """ Return the square of the Euclidean norm of a vector c. """

    return sum([expr^2 for expr in c])

def multc(c, d):
    """ Multiply pairs c and d as though they are complex numbers . """

    return vector([c[0]*d[0] - c[1]*d[1], c[1]*d[0] + c[0]*d[1]])

def factorc(c):
    """ Factor each component of the vector c. """

    return vector([expr.factor() for expr in c])

def dividec(c1, c2):
    """ Divide pairs c1 and c2 as though they are complex numbers . """

    return vector([(c1[0]*c2[0] + c1[1]*c2[1])/(c2[0]^2 + c2[1]^2),␣
 ↪(c1[1]*c2[0] - c1[0]*c2[1])/(c2[0]^2 + c2[1]^2)])

def my_subs(z, D):
    """ Substitute the elements of a vector z by a dictionary D,
    while making sure extra factors get cancelled . """

    return vector([expr.subs(D).factor().expand() for expr in z1])
```

2 Detecting symmetries of space curves

Next we recall code for detecting symmetries of space curves, which was introduced in the paper [2].

[2] Juan Gerardo Alcazar, Carlos Hermoso and Georg Muntingh, *Symmetry Detection of Rational Space Curves from their Curvature and Torsion*, Computer Aided Geometric Design (2015)

[2]:
```
def is_valid(x, verbose = True):
    """ Check whether the curve satisfies the conditions required in the␣
 ↪algorithm in [1]. """
    if prod([component.denominator().subs({t: 0}) for component in x]) == 0:
        if verbose: print("Error. The parametrization is not defined at t = 0.")
        return False
    if curvature2(x).numerator().subs({t: 0}) == 0:
        if verbose: print("Error. The curvature is 0 at t = 0.")
        return False
    if curvature2(x).denominator().subs({t: 0}) == 0:
        if verbose: print("Error. The curvature is not defined at t = 0.")
        return False

    return True

def curvature2(x):
    """ Find the square of the curvature of the parametrization x. """
    return normc2(x.diff(t, 1).cross_product(x.diff(t, 2)))/(normc2( x.diff(t,␣
 ↪1) )^3)

def torsion(x):
    """ Find the torsion of the parametrization x. """
    return (x.diff(t,1).cross_product(x.diff(t,2)).inner_product(x.diff(t,3)))/
 ↪normc2(x.diff(t,1).cross_product(x.diff(t,2)))

def KK(x):
    """ Find the bivariate polynomial K(t, s) in [1] that represents equality␣
 ↪of curvature. """
    curv2 = curvature2(x)
    A, B = curv2.numerator(), curv2.denominator()
    return A*B.subs({t: s}) - A.subs({t: s})*B

def TTpm(x):
    """ Find the bivariate polynomial T^pm (t, s) in [1] that represents␣
 ↪pm-equality of torsion. """
```

```
    tau = torsion(x)
    C, D = tau.numerator(), tau.denominator()
    return C*D.subs({t: s}) - C.subs({t: s})*D, C*D.subs({t: s}) + C.subs({t:␣
↪s})*D

def GGpm(x):
    """ Find the greatest common divisor G(t,s) of K(t,s) and T(t,s) in [1]. """
    K = KK(x)
    Tp, Tm = TTpm(x)

    # The 'modular' algorithm beats the 'ezgcd' algorithm in the examples of␣
↪dense polynomials of high degree.
    gcd_algorithm = 'modular'
    return K.gcd(Tp, algorithm = gcd_algorithm), K.gcd(Tm, algorithm =␣
↪gcd_algorithm)

def MB(x):
    """ Find the matrix B in [1]. """
    return matrix([x.diff(t, 1).subs({t: 0}), x.diff(t, 2).subs({t: 0}), x.
↪diff(t, 1).subs({t: 0}).cross_product(x.diff(t, 2).subs({t: 0}))]).
↪transpose()

def MC0(x, a, b, c, d, detQ):
    """ Find the matrix C in [1] for the case d = 0. """
    ta, tb = b/c, a/c
    xt = x.subs({t:s}).subs({s:1/t})
    return matrix([xt.diff(t,1).subs({t: tb})*ta  ,
                   xt.diff(t,2).subs({t: tb})*ta^2,
                   xt.diff(t,1).subs({t: tb}).cross_product(xt.diff(t,2).
↪subs({t: tb}))*ta^3*detQ ]).transpose()

def MC(x, a, b, c, d, detQ):
    """ Find the matrix C in [1] for the case d != 0. """
    Delta = a*d - b*c
    # print(b, c, Delta, detQ)

    xdiff1 = x.diff(t, 1)
    xdiff2 = x.diff(t, 2)
    xdiff1cross2 = xdiff1.cross_product(xdiff2)
    M1 = vector([xdiff1[i].subs({t: b})*Delta for i in range(3)])
    M2 = vector([xdiff2[i].subs({t: b})*Delta^2 - 2*c*Delta*xdiff1[i].subs({t:␣
↪b}) for i in range(3)])
    # M3 =
    M3 = vector([detQ*Delta^3*xdiff1cross2[i].subs({t: b}) for i in range(3)])
    # print(3, detQ*Delta^3*x.diff(t, 1).subs({t:b}).cross_product(x.diff(t, 2).
↪subs({t:b})))
```

```
    # M = matrix([x.diff(t, 1).subs({t: b})*Delta, \
    #             x.diff(t, 2).subs({t: b})*Delta^2 - 2*c*Delta*x.diff(t, 1).
 ↪subs({t: b}), \
    #             detQ*Delta^3*x.diff(t, 1).subs({t:b}).cross_product(x.diff(t,␣
 ↪2).subs({t:b}))]).transpose()
    M = matrix([M1, M2, M3]).transpose()

    return M

def find_t0(G):
    """ Find an abscissa t0 such that the vertical line at t0 does not
        contain any zero of G where the partial derivative dG/ds vanishes.
    """
    t0 = 2
    g = G.subs({t: t0})
    while g.discriminant(s) == 0:
        t0 += 1
        g = G.subs({t: t0})

    return t0
```

```
[3]: ring.<s,t,a,b,c,d,xi> = PolynomialRing(QQ)

def xiabcd(G, x, t0, coeff_ring=AA, verbose=False):
    """ Find the coefficients a,b,c,d of the Moebius transformations
        phi(t) = (at + b)/(ct + d) corresponding to the symmetries of x
        from the bivariate polynomial G(t,s) and abscissa t0.
    """
    ring.<s,t,a,b,c,d,xi> = PolynomialRing(QQ)

    s0p  = (- diff(G, t)/diff(G, s)).subs({t: t0})
    s0pp = (- (diff(G, t, 2) + 2*diff(diff(G, t), s)*s0p + diff(G, s, 2)*s0p^2)/
 ↪diff(G, s) ).subs({t: t0})

    F = (c*t + d)*s - (a*t + b)
    P = G.resultant(F, s)

    # Case d = 1.
    if 2*s0p + t0*s0pp != 0:
        c0 = -s0pp / (2*s0p + t0*s0pp)
        a0 = s0p + c0*(s + t0*s0p)
        b0 = -a0*t0 + s + c0*t0*s

        R = G.subs({t: t0})
        for i in range(P.degrees()[1]+1):
            R = R.gcd(P.coefficient({t:i}).subs({a: a0, b: b0, c: c0, d: 1}).
↪numerator())

        Lroots = R.univariate_polynomial().roots(ring=coeff_ring)
        Lroots = [ (tup[0], a0.subs({s: tup[0]}), b0.subs({s: tup[0]}), c0.
↪subs({s: tup[0]}), 1) for tup in Lroots]
    else:
        Lroots = []

    # Case d = 0 and c = 1.
    a0 = s + t0*s0p
    b0 = (s - a0)*t0

    R = G.subs({t: t0})
    for i in range(P.degrees()[1]+1):
        R = R.gcd(P.coefficient({t:i}).subs({a: a0, b: b0, c: 1, d: 0}).
↪numerator())

    Lroots0 = R.univariate_polynomial().roots(ring=coeff_ring)
    Lroots0 = [ (tup[0], a0.subs({s: tup[0]}), b0.subs({s: tup[0]}), 1, 0) for␣
↪tup in Lroots0]

    return Lroots + Lroots0

def symmetries(x, coeff_ring=AA, verbose = False):
    """ Find the symmetries of a parametric space curve x. """
    ring.<s,t,a,b,c,d,xi> = PolynomialRing(QQ)

    B = MB(x)
    if verbose: print(f"B = {B}")
    Gpm = GGpm(x)
    for detQ in [0, 1]:
        G = Gpm[detQ]
        if verbose:
            print("\nG^" + (detQ == 0)*"+" + (detQ == 1)*"-" + " =" + str(G.
↪factor()))
        if G != 1:
            t0 = find_t0(G)
            Lxiabcd = xiabcd(G, x, t0, coeff_ring=coeff_ring, verbose = verbose)

            for tup in Lxiabcd:
                a0, b0, c0, d0 = tup[1:]
                if verbose:
                    print("\nSymmetry corresponding to (a,b,c,d) =")
                    print(vector([a0,b0,c0,d0]))
                if d0 == 0:
                    C = MC0(x, a0, b0, c0, d0, (-1)^detQ)
                    Q = C*B.inverse()
                    if prod([component.subs({t: 1/s}).denominator().subs({s: a0/
↪c0}) != 0 for component in x]):
                        bb = x.subs({t: 1/s}).subs({s: a0/c0}) - Q*x.subs({t:␣
↪0})
                    else:
                        print("Error. The parametrization x does not satisfy␣
↪the conditions.")
                else:
                    C  = MC(x, a0, b0, c0, d0, (-1)^detQ)
                    Q  = C*B.inverse()
                    # b0 = SR(b0)
                    # d0 = SR(d0)
                    bb = vector([x[i].subs({t: b0/d0}) for i in range(3)]) -␣
↪Q*x.subs({t: 0})
                    # bb = x.subs({t: b0/d0}) - Q*x.subs({t: 0})

                if verbose:
                    print("C = "); print(C)
                    print("Q = "); print(Q)
                    print("b = " + str(bb))
```

3 Designing translational surfaces with symmetries

Suppose one wishes to design a translational surfaces with certain symmetries. This can be achieved by applying the theory described in [1] as shown in the following examples.

4 Example 2 in [1]: Crunode curve

Let $\mathcal{C}_1 \subset \mathbb{R}^3$ be the crunode curve parametrized by

$$\boldsymbol{c}_1(t) = \big(x(t), y(t), z(t)\big) = \left(\frac{t}{t^4+1}, \frac{t^2}{t^4+1}, \frac{t^3}{t^4+1}\right).$$

This curve is invariant under the half-turn about the y-axis, as

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} x(-t) \\ y(-t) \\ z(-t) \end{bmatrix}.$$

as well as reflections in the planes $z \pm x = 0$, since

$$\begin{bmatrix} 0 & 0 & \pm 1 \\ 0 & 1 & 0 \\ \pm 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} \pm z(t) \\ y(t) \\ \pm x(t) \end{bmatrix} = \begin{bmatrix} x(\pm 1/t) \\ y(\pm 1/t) \\ z(\pm 1/t) \end{bmatrix}.$$

We consider now the curves $\boldsymbol{c}_2$ parametrized as $\boldsymbol{c}_2 = \boldsymbol{c}_1$, $\boldsymbol{c}_2 = \boldsymbol{M}_y \boldsymbol{c}_2$ and $\boldsymbol{c}_2 = \boldsymbol{M}_z \boldsymbol{c}_1$, where the matrices $\boldsymbol{M}_y$ and $\boldsymbol{M}_z$ denote reflections in the planes $y = 0$ and $z = 0$. According to Proposition 4 in [1], in each of these cases the translational surface defined by $\boldsymbol{c}_1 \oplus \boldsymbol{c}_2$ is symmetric.

```
[4]: c1 = vector((t/(t^4 + 1), t^2/(t^4 + 1), t^3/(t^4 + 1)))
     symmetries(c1, verbose=True)
```

```
B = [1 0 0]
[0 2 0]
[0 0 2]

G^+ =(-1) * (s - t) * (s + t)

Symmetry corresponding to (a,b,c,d) =
(-1.000000000000000?, 0.?e-17, 0.?e-19, 1)
C =
[-1  0  0]
[ 0  2  0]
[ 0  0 -2]
Q =
[-1  0  0]
[ 0  1  0]
[ 0  0 -1]
b = (0, 0, 0)

Symmetry corresponding to (a,b,c,d) =
(1.000000000000000?, 0.?e-17, 0.?e-19, 1)
C =
[1 0 0]
[0 2 0]
[0 0 2]
Q =
[1 0 0]
[0 1 0]
[0 0 1]
b = (0, 0, 0)

G^- =(s*t - 1) * (s*t + 1)

Symmetry corresponding to (a,b,c,d) =
(0, -1.000000000000000?, 1, 0)
C =
[ 0  0 -2]
[ 0  2  0]
[-1  0  0]
Q =
[ 0  0 -1]
[ 0  1  0]
[-1  0  0]
b = (0, 0, 0)
```

```
Symmetry corresponding to (a,b,c,d) =
(0, 1.000000000000000?, 1, 0)
C =
[0 0 2]
[0 2 0]
[1 0 0]
Q =
[0 0 1]
[0 1 0]
[1 0 0]
b = (0, 0, 0)
```

[5]:
```
var('u,v')
eps = pi
c1u = vector([c1[i].subs({t: tan(u/2)}) for i in range(len(c1))])
G = parametric_plot3d(c1u, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
 ↪plot_points=100, boundary_style={'thickness':10, 'color': 'red', 'zorder':␣
 ↪10}, frame=False)
show(G)

for c2 in [c1, vector((c1[0],-c1[1],c1[2])), vector((c1[0],c1[1],-c1[2]))]:
    c2v = vector([c2[i].subs({t: tan(v/2)}) for i in range(len(c2))])
    S = c1u + c2v
    G = parametric_plot3d(S, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
 ↪plot_points=[100,100], frame=False, aspect_ratio=1)
    G += parametric_plot3d(c1u - vector((0,0.05,0)), (u, -eps, eps), (v, -eps,␣
 ↪eps), mesh=True, plot_points=100, boundary_style={'thickness':10, 'color':␣
 ↪'red', 'zorder': 10}, frame=False)
    G += parametric_plot3d(c2v - vector((0,0.06,0)), (u, -eps, eps), (v, -eps,␣
 ↪eps), mesh=True, plot_points=100, boundary_style={'thickness':10, 'color':␣
 ↪'green'})

    show(G)
```

```
Graphics3d Object


Graphics3d Object


Graphics3d Object


Graphics3d Object
```

5 Example 2 in [1]: Trefoil knot

Let $\mathcal{C}_1 \subset \mathbb{R}^3$ be the trefoil knot parametrized by

$$\boldsymbol{c}_1(\theta) = \big(x(\theta), y(\theta), z(\theta)\big) := \big(\sin(\theta) + 2\sin(2\theta), \cos(\theta) - 2\cos(2\theta), -\sin(3\theta)\big).$$

This curve is rational, which follows from applying the Weierstrass substitution

$$t = \tan\left(\frac{\theta}{2}\right) \quad \Longleftrightarrow \quad \sin(\theta) = \frac{2t}{1+t^2}, \; \cos(\theta) = \frac{1-t^2}{1+t^2}, \qquad -\pi < \theta < \pi.$$

Using the method from [**?**], one obtains rotational symmetries about the z-axis by angles $\pm 2\pi/3$, as well as a half-turn about the y-axis. Alternatively, these symmetries are determined directly by applying a rotation matrix and standard trigonometric identities. In particular,

$$\begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} x(\theta) \\ y(\theta) \end{bmatrix} = \begin{bmatrix} \sin(\theta - \varphi) + 2\sin(2\theta + \varphi) \\ \cos(\theta - \varphi) - 2\cos(2\theta + \varphi \end{bmatrix} = \begin{bmatrix} x(\theta - \varphi) \\ y(\theta - \varphi) \end{bmatrix}$$

holds identically if and only if $\varphi \equiv 0$ or $\varphi \equiv \pm 2\pi/3$ modulo 2π, while

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x(\theta) \\ y(\theta) \\ z(\theta) \end{bmatrix} = \begin{bmatrix} x(-\theta) \\ y(-\theta) \\ z(-\theta) \end{bmatrix}.$$

Choosing $\boldsymbol{c}_2 = \boldsymbol{c}_1$, $\boldsymbol{c}_2 = \boldsymbol{M}_y \boldsymbol{c}_2$ and $\boldsymbol{c}_2 = \boldsymbol{M}_z \boldsymbol{c}_1$ as above yields symmetric translational surfaces defined by $\boldsymbol{c}_1 \oplus \boldsymbol{c}_2$.

[6]:
```
s0 = 2*t/(1 + t^2)
c0 = (1 - t^2)/(1 + t^2)
c1t = vector((s0 + 4*s0*c0, c0 - 2*(c0^2 - s0^2), -(3*s0 - 4*s0^3)))
symmetries(c1t, verbose=True)

s0 = sin(t)
c0 = cos(t)
c1 = vector((s0 + 4*s0*c0, c0 - 2*(c0^2 - s0^2), -(3*s0 - 4*s0^3)))
# c1 = vector((sin(t) + 2*sin(2*t), cos(t) - 2*cos(2*t), -sin(3*t)))
c1u = vector([c1[i].subs({t: u}) for i in range(len(c1))])
```

```
B = [ 10   0 168]
[  0  28   0]
[ -6   0 280]

G^+ =(-s + t) * (s + t) * (3*s^2*t^2 - s^2 - 8*s*t - t^2 + 3) * (3*s^2*t^2 - s^2
+ 8*s*t - t^2 + 3)

Symmetry corresponding to (a,b,c,d) =
(-1.000000000000000?, 0.?e-13, 0.?e-14, 1)
C =
[ -10    0 -168]
[   0   28    0]
[   6    0 -280]
Q =
[-1  0  0]
[ 0  1  0]
[ 0  0 -1]
b = (0, 0, 0)

Symmetry corresponding to (a,b,c,d) =
(1.0000000000000?, 1.732050807569?, -1.7320508075689?, 1)
C =
[                 -5  24.24871130596429?                 -84]
[ -8.66025403784439?                 -14 -145.4922678357857?]
[                 -6                   0                 280]
Q =
[               -1/2  0.866025403784439?                   0]
[-0.866025403784439?                -1/2                   0]
[                  0                   0                   1]
b = (0, 0, 0)

Symmetry corresponding to (a,b,c,d) =
(-1.000000000000000?, 1.732050807568878?, 1.732050807568878?, 1)
C =
[                 5 24.24871130596429?                 84]
[ 8.66025403784439?                -14 145.4922678357857?]
[                 6                  0               -280]
Q =
[               1/2 0.866025403784439?                  0]
[0.866025403784439?               -1/2                  0]
[                 0                  0                 -1]
b = (0, 0, 0)

Symmetry corresponding to (a,b,c,d) =
(1.000000000000000?, -1.732050807568878?, 1.732050807568878?, 1)
C =
[                 -5 -24.24871130596429?                 -84]
[  8.66025403784439?                 -14  145.4922678357857?]
[                 -6                   0                 280]
Q =
[               -1/2 -0.866025403784439?                   0]
[ 0.866025403784439?                -1/2                   0]
[                  0                   0                   1]
b = (0, 0, 0)

Symmetry corresponding to (a,b,c,d) =
(-1.0000000000000?, -1.7320508075689?, -1.7320508075689?, 1)
C =
[                  5 -24.24871130596429?                  84]
[ -8.66025403784439?                 -14 -145.4922678357857?]
[                   6                    0                -280]
Q =
[                 1/2 -0.866025403784439?                    0]
[-0.866025403784439?                 -1/2                    0]
[                   0                    0                   -1]
b = (0, 0, 0)

Symmetry corresponding to (a,b,c,d) =
(1.000000000000?, 0.?e-13, 0.?e-14, 1)
C =
[ 10   0 168]
[  0  28   0]
[ -6   0 280]
Q =
[1 0 0]
[0 1 0]
[0 0 1]
b = (0, 0, 0)

G^- =1
```

[7]:
```
var('u,v')
n = 2
eps = pi
c2v = vector([(-1)^(i+1)*c1[i].subs({t: v}) for i in range(len(c1))])
G = parametric_plot3d(c1u - vector((0,0.05,0)), (u, -eps, eps), (v, -eps, eps),␣
↪mesh=True, plot_points=100, boundary_style={'thickness':10, 'color': 'red',␣
↪'zorder': 10}, frame=True)
G += parametric_plot3d(c2v - vector((0,0.06,0)), (u, -eps, eps), (v, -eps,␣
↪eps), mesh=True, plot_points=100, boundary_style={'thickness':10, 'color':␣
↪'green', 'zorder': 10}, frame=True)
show(G)
for c2 in [c1, vector((c1[0],-c1[1],c1[2])), vector((c1[0],c1[1],-c1[2]))]:
    c2v = vector([c2[i].subs({t: v}) for i in range(len(c2))])
    S = c1u + c2v
    G = parametric_plot3d(S, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
↪plot_points=[100,100], frame=False, aspect_ratio=1)
    G += parametric_plot3d(c1u - vector((0,0.05,0)), (u, -eps, eps), (v, -eps,␣
↪eps), mesh=True, plot_points=100, boundary_style={'thickness':10, 'color':␣
↪'red', 'zorder': 10}, frame=False)
    G += parametric_plot3d(c2v - vector((0,0.06,0)), (u, -eps, eps), (v, -eps,␣
↪eps), mesh=True, plot_points=100, boundary_style={'thickness':10, 'color':␣
↪'green'})

    show(G)
```

```
Graphics3d Object
```

```
Graphics3d Object

Graphics3d Object

Graphics3d Object
```

6 Example: Translational surfaces generated by Rose curves

Consider the family of planar Rose curves of degree $2n+2$ parametrized by

$$\mathcal{C}^n : t \longmapsto \big(u_n(t), v_n(t)\big) := w_n(t)\cdot(2t, 1-t^2) \in \mathbb{R}^2, \qquad w_n(t) = \frac{1}{(1+t^2)^{n+1}} \sum_{k=0}^{n} \binom{2n}{2k} (-1)^k t^{2k}, \qquad n = 1,2,3,\ldots,$$

as well as the two (degenerate) parametric space curve embeddings

$$\mathcal{C}_1^n : t \longmapsto c_1^n(t) := \big(u_n(t), v_n(t), 0\big) \in \mathbb{R}^3,$$
$$\mathcal{C}_2^n : t \longmapsto c_2^n(t) := \big(u_n(t), 0, v_n(t)\big) \in \mathbb{R}^3.$$

Clearly these curves are related by an isometry

$$c_1^n(t) = \boldsymbol{P} c_2^n(t), \qquad \boldsymbol{P} := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

For $n \geq 2$, the Rose curve $\mathcal{C}^n$ has the same symmetries as a regular polygon with n vertices. Hence its symmetry group is the corresponding dihedral group.

[8]:
```
var('u,v')
n = 2
d = sum([binomial(2*n,2*k)*(-1)^k * t^(2*k) for k in range(n+1)])

c1 = vector((2*t, 1-t^2, 0))* d / (1 + t^2)^(n+1)
# symmetries(c1, verbose=True)
c2 = vector((c1[0], c1[2], c1[1]))

eps = pi
c1u = vector([c1[i].subs({t: tan(u/2)}) for i in range(len(c1))])
c2v = vector([c2[i].subs({t: tan(v/2)}) for i in range(len(c2))])
S = c1u + c2v
G = parametric_plot3d(S, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
 ↪plot_points=[100,100], frame=False, aspect_ratio=1)
G += parametric_plot3d(c1u, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
 ↪plot_points=100, boundary_style={'thickness':10, 'color': 'red'})
```

```
G += parametric_plot3d(c2v, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
↪plot_points=100, boundary_style={'thickness':10, 'color': 'green'})

show(G)
```

```
Graphics3d Object
```

7 Daisies

Consider the family of *daisies* of increasing degree $m = 4j + 4$, which are given parametrically by

$$\boldsymbol{x}(t) = \left(u \sum_{i=0}^{j} (-1)^i \binom{2j}{2i} u^{2j-2i} v^{2i}, v \sum_{i=0}^{j} (-1)^i \binom{2j}{2i} u^{2j-2i} v^{2i}, \frac{1 - t^{4j+4}}{1 + t^{4j+4}} \right),$$

with

$$u = \frac{1-t^2}{1+t^2}, \qquad v = \frac{2t}{1+t^2}, \qquad j = 0, 1, \ldots$$

```
[9]: var('u,v')
j = 1
u0 = (1-t^2)/(1+t^2)
v0 = 2*t/(1+t^2)
s0 = sum([(-1)^i*binomial(2*j,2*i)*u0^(2*j-2*i)*v0^(2*i) for i in range(j+1)])
c1 = vector((u0*s0, v0*s0, (1-t^(4*j+4))/(1+t^(4*j+4))))
symmetries(c1, verbose=True)
c2 = vector((c1[0], c1[1], -c1[2]))

eps = pi
c1u = vector([c1[i].subs({t: tan(u/2)}) for i in range(len(c1))])
c2v = vector([c2[i].subs({t: tan(v/2)}) for i in range(len(c2))])
S = c1u + c2v
G = parametric_plot3d(S, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
↪plot_points=[100,100], frame=False, aspect_ratio=1)
G += parametric_plot3d(c1u, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
↪plot_points=100, boundary_style={'thickness':10, 'color': 'red'})
G += parametric_plot3d(c2v, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
↪plot_points=100, boundary_style={'thickness':10, 'color': 'green'})

show(G)
```

```
B = [  0 -20   0]
[  2   0   0]
[  0   0  40]

G^+ =(-s + t) * (s*t - 1)
```

```
Symmetry corresponding to (a,b,c,d) =
(1, 0, 0, 1)
C =
[  0 -20   0]
[  2   0   0]
[  0   0  40]
Q =
[1 0 0]
[0 1 0]
[0 0 1]
b = (0, 0, 0)

Symmetry corresponding to (a,b,c,d) =
(0, 1, 1, 0)
C =
[  0  20   0]
[  2   0   0]
[  0   0 -40]
Q =
[-1  0  0]
[ 0  1  0]
[ 0  0 -1]
b = (0, 0, 0)

G^- =(s + t) * (s*t + 1)

Symmetry corresponding to (a,b,c,d) =
(-1, 0, 0, 1)
C =
[  0 -20   0]
[ -2   0   0]
[  0   0  40]
Q =
[ 1  0  0]
[ 0 -1  0]
[ 0  0  1]
b = (0, 0, 0)

Symmetry corresponding to (a,b,c,d) =
(0, -1, 1, 0)
C =
[  0  20   0]
[ -2   0   0]
[  0   0 -40]
Q =
[-1  0  0]
[ 0 -1  0]
[ 0  0 -1]
```

```
b = (0, 0, 0)
```

```
Graphics3d Object
```

8 Detection of affine equivalence

Make an example based on generating highly symmetric surfaces from highly symmetric space curve, just using the same curve for C1 and C2, and one related by a simple rigid motion respecting the symmetries.

[10]:
```
def apply_alpha(alpha, t):
    return (alpha[0][0]*t + alpha[0][1])/(alpha[1][0]*t + alpha[1][1])
    # return (alpha[1][0] + alpha[1][1]*t)/(alpha[0][0] + alpha[0][1]*t)


def degreec(p):
    return max([expr.degree() for expr in p])


def coefficientsc(p):
    # Find the coefficient vectors of a parametrization p
    return [vector([expr.coefficient({t:i}) for expr in p]) for i in␣
 ↪range(degreec(p)+1)]


def compose_alphac(p, alpha):
    return vector([expr.subs({t: apply_alpha(alpha, t)}) for expr in p])


def numeratorc(p):
    p = lcm([expr.denominator() for expr in p])*p
    return vector([expr.numerator() for expr in p])


def reparametrized_coeffs(cs, a, method=0):
    if method == 0:
        d = len(cs) - 1
        return␣
 ↪[sum([cs[i]*sum([binomial(d-i,l)*binomial(i,j-l)*a[0][0]^(d-i-l)*a[0][1]^l*a[1][0]^(i-j+l)*
                              for l in range(j+1) if l <= d - i and j - l <=␣
 ↪i]) for i in range(d+1)]) for j in range(d+1)]
    elif method == 1:
        p = sum([cs[i] * t^i for i in range(len(cs))])
        p = compose_alphac(p, a)
        p = numeratorc(p)
        return coefficientsc(p)
```

```
def find_eqs(c, d, is_orthogonal=True, verbose=False, affine_term=True, eqs=[]):
    A = matrix([[a00, a01], [a10, a11]])
    M = matrix([[m00, m01, m02],[m10, m11, m12],[m20, m21, m22]])

    if is_orthogonal:
        MtM = M.transpose()*M
        eqs = eqs + [MtM[i][j] - (i==j) for i in range(3) for j in range(3)]

    if affine_term:
        b = matrix([[b0], [b1], [b2]])
        lhs = (M*c.subs({s: 1}) + vector([b0, b1, b2])) * (a10*t + a11)^d.
 ↪degree()
    else:
        lhs = M*c.subs({s: 1}) * (a10*t + a11)^d.degree()

    rhs = d.subs({t: a00*t + a01, s: a10*t + a11})
    for i in range(3):
        expr = lhs[i] - rhs[i]
        eqs = eqs + [expr.coefficient({t: j}) for j in range(expr.degree(t)+1)]

    return eqs


def show_solutions(groebner, affine_term=False):
    var('latexfx', latex_name='\\mathbf{f}(\\mathbf{x})')
    var('latexx', latex_name='\\mathbf{x}')
    var('latexphit', latex_name='\\qquad \\varphi(t)')

    if affine_term:
        ring0.<a00,a01,a10,a11,m00,m01,m02,m10,m11,m12,m20,m21,m22,b0,b1,b2> =␣
 ↪PolynomialRing(QQ)
    else:
        ring0.<a00,a01,a10,a11,m00,m01,m02,m10,m11,m12,m20,m21,m22> =␣
 ↪PolynomialRing(QQ)

    J = ring0.ideal(groebner, coerce=True)
    sols = J.variety()
    for sol in sols:
        M = matrix([[m00, m01, m02],[m10, m11, m12],[m20, m21, m22]])
        M0 = matrix([[expr.subs(sol) for expr in row] for row in M])

        A = matrix([[a00, a01], [a10, a11]])
        A0 = matrix([[expr.subs(sol) for expr in row] for row in A])

        if affine_term:
            B = matrix([[b0], [b1], [b2]])
            B0 = matrix([[expr.subs(sol) for expr in row] for row in B])
            show(latexfx," = ", M0, latexx, " + ", B0, "    ", latexphit, " =␣
↪", (A0[0][0]*t + A0[0][1])/(A0[1][0]*t + A0[1][1]) )
        else:
            show(latexfx," = ", M0, latexx, "    ", latexphit, " = ",␣
↪(A0[0][0]*t + A0[0][1])/(A0[1][0]*t + A0[1][1]) )
```

9 Example 1 in [1]: Twisted cubic translational surfaces

Consider the twisted cubic curves

$$\mathcal{C} : t \longmapsto \boldsymbol{c}(t) = \big(t, t^2, t^3\big), \qquad \mathcal{D} : t \longmapsto \boldsymbol{d}(t) = \big(t^3, -t, t^2\big),$$

as well as corresponding surfaces of translation $\mathcal{C} \oplus \mathcal{C}$ and $\mathcal{D} \oplus \mathcal{D}$ obtained by translating these curves along themselves. In this case Algorithm "affine-equiv-trans" in [1] simplifies, as it is only requires detecting affine equivalences $\boldsymbol{f} : \mathcal{C} \longrightarrow \mathcal{C}$.

For ease of presentation we will restrict our attention to affine equivalences $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{M}\boldsymbol{x} + \boldsymbol{b}$ with zero constant term, i.e., $\boldsymbol{b} = 0$.

[11]:
```
affine_term = False
is_orthogonal = False
verbose = True

if affine_term:
    ring.<s,t,a00,a01,a10,a11,m00,m01,m02,m10,m11,m12,m20,m21,m22,b0,b1,b2> =␣
↪PolynomialRing(QQ)
else:
    ring.<s,t,a00,a01,a10,a11,m00,m01,m02,m10,m11,m12,m20,m21,m22> =␣
↪PolynomialRing(QQ)

var('u v')

# Twisted cubic
c1 = vector((t*s^2, t^2 *s, t^3))
c2 = vector((t*s^2, t^2 *s, t^3))
d1 = vector((t^3, -t*s^2, t^2*s))
d2 = vector((t^3, -t*s^2, t^2*s))

c1u = vector([c1[i].subs({t: u, s: 1}) for i in range(len(c1))])
c2v = vector([c2[i].subs({t: v, s: 1}) for i in range(len(c2))])
d1u = vector([d1[i].subs({t: u, s: 1}) for i in range(len(d1))])
d2v = vector([d2[i].subs({t: v, s: 1}) for i in range(len(d2))])

eps = 1
Sc = c1u + c2v
Sd = d1u + d2v
```

```
G = parametric_plot3d(Sc, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
↪plot_points=[50,50], frame=False, aspect_ratio=1)
G += parametric_plot3d(Sd, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
↪plot_points=[50,50], frame=False, aspect_ratio=1, color='red')
G += parametric_plot3d(c1u, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
↪plot_points=100, boundary_style={'thickness':10, 'color': 'yellow', 'zorder':
↪ 10}, frame=False)
G += parametric_plot3d(d1u, (u, -eps, eps), (v, -eps, eps), mesh=True,␣
↪plot_points=100, boundary_style={'thickness':10, 'color': 'green', 'zorder':␣
↪10}, frame=False)
show(G)
```

```
Graphics3d Object
```

We illustrate how the isometries between the generators are obtained, which give rise to isometries between the surfaces.

[12]:
```
for alpha in [1, -1]:
    eqs = [a11 - 1, a00 - alpha]  # [a11, -a10*a01-1] or [a00*a11 - a01*a10 -␣
↪1] or [a00*a11 - a01*a10 + 1]
    eqs = find_eqs(c1, d1, eqs=eqs, is_orthogonal=is_orthogonal,␣
↪affine_term=affine_term, verbose=verbose)
    if verbose:
        print("\nFound the following equations:")
        for eq in eqs:
            print(eq)

    groebner = ring.ideal(eqs).groebner_basis()
    I = ring.ideal(groebner)
    if verbose:
        print("\n... generating an ideal with Groebner basis is generated by")
        for expr in groebner:
            print(expr)

    show_solutions(groebner, affine_term=affine_term)
```

```
Found the following equations:
a11 - 1
a00 - 1
-a01^3
a11^3*m00 - 3*a00*a01^2
3*a10*a11^2*m00 + a11^3*m01 - 3*a00^2*a01
3*a10^2*a11*m00 + 3*a10*a11^2*m01 + a11^3*m02 - a00^3
a10^3*m00 + 3*a10^2*a11*m01 + 3*a10*a11^2*m02
a10^3*m01 + 3*a10^2*a11*m02
a10^3*m02
```

```
a01*a11^2
a11^3*m10 + 2*a01*a10*a11 + a00*a11^2
3*a10*a11^2*m10 + a11^3*m11 + a01*a10^2 + 2*a00*a10*a11
3*a10^2*a11*m10 + 3*a10*a11^2*m11 + a11^3*m12 + a00*a10^2
a10^3*m10 + 3*a10^2*a11*m11 + 3*a10*a11^2*m12
a10^3*m11 + 3*a10^2*a11*m12
a10^3*m12
-a01^2*a11
a11^3*m20 - a01^2*a10 - 2*a00*a01*a11
3*a10*a11^2*m20 + a11^3*m21 - 2*a00*a01*a10 - a00^2*a11
3*a10^2*a11*m20 + 3*a10*a11^2*m21 + a11^3*m22 - a00^2*a10
a10^3*m20 + 3*a10^2*a11*m21 + 3*a10*a11^2*m22
a10^3*m21 + 3*a10^2*a11*m22
a10^3*m22

… generating an ideal with Groebner basis is generated by
a00 - 1
a01
a10
a11 - 1
m00
m01
m02 - 1
m10 + 1
m11
m12
m20
m21 - 1
m22
latexfx ' = ' [ 0  0  1]
[-1  0  0]
[ 0  1  0] latexx '    ' latexphit ' = ' t



Found the following equations:
a11 - 1
a00 + 1
-a01^3
a11^3*m00 - 3*a00*a01^2
3*a10*a11^2*m00 + a11^3*m01 - 3*a00^2*a01
3*a10^2*a11*m00 + 3*a10*a11^2*m01 + a11^3*m02 - a00^3
a10^3*m00 + 3*a10^2*a11*m01 + 3*a10*a11^2*m02
a10^3*m01 + 3*a10^2*a11*m02
a10^3*m02
a01*a11^2
a11^3*m10 + 2*a01*a10*a11 + a00*a11^2
3*a10*a11^2*m10 + a11^3*m11 + a01*a10^2 + 2*a00*a10*a11
```

```
3*a10^2*a11*m10 + 3*a10*a11^2*m11 + a11^3*m12 + a00*a10^2
a10^3*m10 + 3*a10^2*a11*m11 + 3*a10*a11^2*m12
a10^3*m11 + 3*a10^2*a11*m12
a10^3*m12
-a01^2*a11
a11^3*m20 - a01^2*a10 - 2*a00*a01*a11
3*a10*a11^2*m20 + a11^3*m21 - 2*a00*a01*a10 - a00^2*a11
3*a10^2*a11*m20 + 3*a10*a11^2*m21 + a11^3*m22 - a00^2*a10
a10^3*m20 + 3*a10^2*a11*m21 + 3*a10*a11^2*m22
a10^3*m21 + 3*a10^2*a11*m22
a10^3*m22

… generating an ideal with Groebner basis is generated by
a00 + 1
a01
a10
a11 - 1
m00
m01
m02 + 1
m10 - 1
m11
m12
m20
m21 - 1
m22

latexfx ' = ' [ 0  0 -1]
[ 1  0  0]
[ 0  1  0] latexx '    ' latexphit ' = ' -t
```