Computational Geometry Workshop

Wichita State University, Dept. of Mathematics

Surfaces of Revolution

Last changed: 24 September 2017

Author: Justin M. Ryan

In [1]:
```
version()
```

Out[1]:
```
'SageMath version 8.0, Release Date: 2017-07-21'
```

In [2]:
```
%display latex
viewer3d = 'jmol'
```

We begin by defining python functions that return the parametrization and chart of the associated surface of revolution.

In [3]:
```
def param(func): #parametrization map from (u,v) plane to surface
   F = func;
   var('u v')
   surf = vector([u,F(x = u)*cos(v),F(x = u)*sin(v)]);
   return surf

def chart(X,Y,Z): #chart from surface to (u,v) plane
   CH = {};
   var('x');
   S(x) = arcsin(x);
   C(x) = arccos(x);
   ff = {S(x),C(x)};
   qq = (Y/sqrt(Y^2 + Z^2));
   for i in range(2):
      CH[i] = vector([X,ff[i](x = qq)]);
   return CH
```

In [4]:
```
var('x');
var('t');
f(x) = 2 + cos(x);
Sf = param(f)
```

In [5]:
```
Sf
```

Out[5]: $(u,\ (\cos(u) + 2)\cos(v),\ (\cos(u) + 2)\sin(v))$

In [6]:
```
parametric_plot3d(Sf,(u,-2*pi,2*pi),(v,0,2*pi))
```

Out[6]:

Let's look at some plot options.

In [7]:
```
surf_plot = parametric_plot3d(Sf,(u,-2*pi,2*pi),(v,0,2*pi),opacity=0.5);
original_curve = parametric_plot3d([t,f(t),0],(t,-2*pi,2*pi),thickness=2,color='red');
show(surf_plot + original_curve)
```

In [8]:
```
%display latex
var('t');
t1(t) = t^3;
t2(t) = exp(t);
curve(t) = [Sf[0](u=t1,v=t2),Sf[1](u=t1,v=t2),Sf[2](u=t1,v=t2)];
curve_plot = parametric_plot3d(curve,(t,-2,2),color='red',thickness=2);
curve
```

Out[8]: $$t \mapsto \left(t^3,\ \left(\cos\left(t^3\right) + 2\right)\cos\left(e^t\right),\ \left(\cos\left(t^3\right) + 2\right)\sin\left(e^t\right)\right)$$

In [9]:
```
curve_plot
```

Out[9]:

This curve should lie on the surface.

In [10]:
```
show(surf_plot + curve_plot)
```

Next: induced metric, Christoffel symbols, and geodesics.

In [11]:
```
def sr_metric(func):
    F = func;
    FF = param(F);
    uu = vector([FF[0].diff(u),FF[1].diff(u),FF[2].diff(u)]);
    vv = vector([FF[0].diff(v),FF[1].diff(v),FF[2].diff(v)]);
    g11 = uu.dot_product(uu).simplify_full();
    g12 = uu.dot_product(vv).simplify_full();
    g22 = vv.dot_product(vv).simplify_full();
    M = matrix([[g11,g12],[g12,g22]]);
    return M
```

In [12]:
```
g = sr_metric(f);
g
```

Out[12]:
$$\begin{pmatrix} \sin(u)^2 + 1 & 0 \\ 0 & \cos(u)^2 + 4\cos(u) + 4 \end{pmatrix}$$

In [13]:
```
def sr_christoffel(func):
  F = func;
  G = sr_metric(F);
  GI = G.inverse();
  GG = [None]*2;
  vv = [u,v];
  for i in range(2):
    GG[i] = {}
    for j in range(2):
      GG[i][j] = {}
      for k in range(2):
        GG[i][j][k] = (1/2)*sum(GI[k,m]*(G[m,i].diff(vv[j]) +
                        G[m,j].diff(vv[i]) - G[i,j].diff(vv[m])) for m in range(2)).simplify_full();
  return GG
```

In [14]:
```
CS = sr_christoffel(f);
```

In [15]:
```
for i in range(2):
  for j in range(2):
    for k in range(2):
      print (i,j,k), ' : ', CS[i][j][k]
```

```
(0, 0, 0)  :  cos(u)*sin(u)/(sin(u)^2 + 1)
(0, 0, 1)  :  0
(0, 1, 0)  :  0
(0, 1, 1)  :  -sin(u)/(cos(u) + 2)
(1, 0, 0)  :  0
(1, 0, 1)  :  -sin(u)/(cos(u) + 2)
(1, 1, 0)  :  (cos(u) + 2)*sin(u)/(sin(u)^2 + 1)
(1, 1, 1)  :  0
```

The Christoffel symbols are the coefficients of the Levi-Civita connection (or covariant derivative), which describes how tangent vectors are parallel transported along curves in the surface. If $\{\partial_1, \ldots \partial_n\}$ are the basis vector fields induced by the local coordinates, then

$$\nabla_{\partial_i} \partial_j = \Sigma_{k=1}^{n} \Gamma_{ij}^{k} \partial_k .$$

In particular, this applies to the velocity vector field of a curve along the curve itself: A curve $\alpha : (-t_0, t_0) \to S_f$ is said to be a *geodesic* if its velocity vector field is parallel; that is, if

$$\ddot{\alpha} := \nabla_{\dot{\alpha}} \dot{\alpha} = 0.$$

It is important to note here that $\ddot{\alpha}$ is not the component-wise second derivative from Calculus III. It is the covariant derivative of the velocity field along itself. The covariant derivative can be written out in terms of its coefficients, the Christoffel symbols. Thus we obtain a system of differential equations that determine criteria for α to be a geodesic. These are known as the geodesic equation(s). If $\alpha(t) = [\alpha_0(t), \alpha_1(t)]$ (indices chosen to match Sage code), then the geodesic equation for the k^{th} component is

$$\frac{d^2 \alpha_k}{dt^2} + \Sigma_{i,j=0}^{1} \Gamma_{ij}^{k} \frac{d\alpha_i}{dt} \frac{d\alpha_j}{dt} = 0.$$

In [16]:
```
def sr_geodesic(func,pos,vel): # geodesic equation
    F = func;
    # pos is initial position
    # vel is initial velocity
    CSF = sr_christoffel(F);
    t = var('t');
    U = function('U')(t);
    V = function('V')(t);
    WW = U.diff(t);
    ZZ = V.diff(t)
    comp = [U,V,WW,ZZ];
    eqn = {}
    for k in range(2):
        eqn[k] = (comp[k+2].diff(t) + sum(sum(CSF[i][j][k](u = U)(v = V)*comp[i].diff(t)*comp[j].diff(t
) for i in range(2)) for j in range(2)) == 0);
    Sol = desolve_system([eqn[0],eqn[1]],[U,V,WW,ZZ],ivar=t,ics=[0,pos[0],pos[1],vel[0],vel[1]])
    UU = Sol[0].simplify_full();
    VV = Sol[1].simplify_full();
    return UU, VV
```

In [17]:
```
GEO = sr_geodesic(f,[0,0],[0,1])
```

In [18]:
```
GEO
```

Out[18]:

$$\left\{ U(t) = \mathcal{L}^{-1}\left[-\frac{\mathcal{L}\left(\frac{\cos(U(t))\,\sin(U(t))\,\frac{\partial}{\partial t}U(t)^2}{\sin(U(t))^2+1}, t, g_{2835} \right) + \mathcal{L}\left(\frac{\cos(U(t))\,\sin(U(t))}{\sin(U(t))^2+} \right.}{g_{2835}^2} \right.$$

$$\left. V(t) = \mathcal{L}^{-1}\left[\frac{2\,\mathcal{L}\left(\frac{\sin(U(t))\,\frac{\partial}{\partial t}U(t)\,\frac{\partial}{\partial t}V(t)}{\cos(U(t))+2}, \right.}{g_{2835}^2} \right.$$

In [19]:

```
surf_plot = parametric_plot3d(Sf,(u,-2*pi,2*pi),(v,0,2*pi),opacity=0.25);
geod_plot = parametric_plot3d([Sf[0](u = GEO[0].rhs())(v = GEO[1].rhs()),
                               Sf[1](u = GEO[0].rhs())(v = GEO[1].rhs()),
                               Sf[2](u = GEO[0].rhs())(v = GEO[1].rhs())],(t,-4,4),color='red',thickness=1.5);
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-8e9b5d134a5a> in <module>()
      2 geod_plot = parametric_plot3d([Sf[Integer(0)](u = GEO[Integer(0)].rhs())(v = GEO[Integer(1)
].rhs()),
      3                                Sf[Integer(1)](u = GEO[Integer(0)].rhs())(v = GEO[Integer(1)].rhs()),
----> 4                                 Sf[Integer(2)](u = GEO[Integer(0)].rhs())(v = GEO[Integer(1)].rhs())],(t,-Inte
ger(4),Integer(4)),color='red',thickness=RealNumber('1.5'));

/home/justin/bin/Sage/SageMath/local/lib/python2.7/site-packages/sage/plot/plot3d/parametric
_plot3d.pyc in parametric_plot3d(f, urange, vrange, plot_points, boundary_style, **kwds)
  1004       if plot_points == "automatic":
  1005           plot_points = 75
-> 1006       G = _parametric_plot3d_curve(f, urange, plot_points=plot_points, **kwds)
  1007     else:
  1008       if plot_points == "automatic":

/home/justin/bin/Sage/SageMath/local/lib/python2.7/site-packages/sage/plot/plot3d/parametric
_plot3d.pyc in _parametric_plot3d_curve(f, urange, plot_points, **kwds)
  1061     """
  1062     from sage.plot.misc import setup_for_eval_on_grid
-> 1063     g, ranges = setup_for_eval_on_grid(f, [urange], plot_points)
  1064     f_x, f_y, f_z = g
  1065     w = [(f_x(u), f_y(u), f_z(u)) for u in xsrange(*ranges[0], include_endpoint=True)]

/home/justin/bin/Sage/SageMath/local/lib/python2.7/site-packages/sage/plot/misc.pyc in setup_f
or_eval_on_grid(funcs, ranges, plot_points, return_vars)
   148       return fast_float(funcs, *vars,**options), [tuple(range+[range_step]) for range,range_s
tep in zip(ranges, range_steps)], vars
   149     else:
--> 150       return fast_float(funcs, *vars,**options), [tuple(range+[range_step]) for range,range_
step in zip(ranges, range_steps)]
   151
   152

/home/justin/bin/Sage/SageMath/src/sage/ext/fast_eval.pyx in sage.ext.fast_eval.fast_float (/ho
me/justin/bin/Sage/SageMath/src/build/cythonized/sage/ext/fast_eval.c:10920)()
  1381
  1382     if isinstance(f, (tuple, list)):
-> 1383         return tuple([fast_float(x, *vars, expect_one_var=expect_one_var) for x in f])
  1384
  1385     cdef int i

/home/justin/bin/Sage/SageMath/src/sage/ext/fast_eval.pyx in sage.ext.fast_eval.fast_float (/ho
me/justin/bin/Sage/SageMath/src/build/cythonized/sage/ext/fast_eval.c:11141)()
  1396           return f._fast_float_(*vars)
  1397       else:
-> 1398           return fast_callable(f, vars=vars, domain=float, _autocompute_vars_for_backward_c
ompatibility_with_deprecated_fast_float_functionality=True, expect_one_var=expect_one_var)
  1399     except AttributeError:
  1400       pass
```

Let's try a "nicer" function.

In [20]:
```
g(x) = 2;
Sg = param(g);
geo_g1 = sr_geodesic(g,[0,0],[1,0]);
geo_g2 = sr_geodesic(g,[0,0],[0,1]);
geo_g3 = sr_geodesic(g,[0,0],[1,1]);
```

In [21]:
```
geo_plot1 = parametric_plot3d([Sg[0](u = geo_g1[0].rhs())(v = geo_g1[1].rhs()),
                    Sg[1](u = geo_g1[0].rhs())(v = geo_g1[1].rhs()),
                    Sg[2](u = geo_g1[0].rhs())(v = geo_g1[1].rhs())],(t,-4,4),color='red',thickness=1.5);
geo_plot2 = parametric_plot3d([Sg[0](u = geo_g2[0].rhs())(v = geo_g2[1].rhs()),
                    Sg[1](u = geo_g2[0].rhs())(v = geo_g2[1].rhs()),
                    Sg[2](u = geo_g2[0].rhs())(v = geo_g2[1].rhs())],(t,-4,4),color='black',thickness=1.
5);
geo_plot3 = parametric_plot3d([Sg[0](u = geo_g3[0].rhs())(v = geo_g3[1].rhs()),
                    Sg[1](u = geo_g3[0].rhs())(v = geo_g3[1].rhs()),
                    Sg[2](u = geo_g3[0].rhs())(v = geo_g3[1].rhs())],(t,-4,4),color='purple',thickness=1
.5);
surf_plot = parametric_plot3d(Sg,(u,-4,4),(v,0,2*pi),opacity=0.25);
show(surf_plot + geo_plot1 + geo_plot2 + geo_plot3)
```

In [22]:
```
h(x) = 1+ x^2;
Sh = param(h);
geo_h1 = sr_geodesic(h,[0,0],[1,0]);
geo_h2 = sr_geodesic(h,[0,0],[0,1]);
geo_h3 = sr_geodesic(h,[0,0],[1,1]);
```

In [23]:
```
geo_hplot1 = parametric_plot3d([Sh[0](u = geo_h1[0].rhs())(v = geo_h1[1].rhs()),
                    Sh[1](u = geo_h1[0].rhs())(v = geo_h1[1].rhs()),
                    Sh[2](u = geo_h1[0].rhs())(v = geo_h1[1].rhs())],(t,-4,4),color='red',thickness=1.5);
geo_hplot2 = parametric_plot3d([Sh[0](u = geo_h2[0].rhs())(v = geo_h2[1].rhs()),
                    Sh[1](u = geo_h2[0].rhs())(v = geo_h2[1].rhs()),
                    Sh[2](u = geo_h2[0].rhs())(v = geo_h2[1].rhs())],(t,-4,4),color='black',thickness=1.
5);
geo_hplot3 = parametric_plot3d([Sh[0](u = geo_h3[0].rhs())(v = geo_h3[1].rhs()),
                    Sh[1](u = geo_h3[0].rhs())(v = geo_h3[1].rhs()),
                    Sh[2](u = geo_h3[0].rhs())(v = geo_h3[1].rhs())],(t,-4,4),color='purple',thickness=1
.5);
surf_plot = parametric_plot3d(Sh,(u,-4,4),(v,0,2*pi),opacity=0.25);
show(surf_plot + geo_hplot1 + geo_hplot2 + geo_hplot3)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-23-79c0a0998c06> in <module>()
      1 geo_hplot1 = parametric_plot3d([Sh[Integer(0)](u = geo_h1[Integer(0)].rhs())(v = geo_h1[Int
eger(1)].rhs()),
      2                                 Sh[Integer(1)](u = geo_h1[Integer(0)].rhs())(v = geo_h1[Integer(1)].rhs()),
----> 3                                  Sh[Integer(2)](u = geo_h1[Integer(0)].rhs())(v = geo_h1[Integer(1)].rhs())],(
t,-Integer(4),Integer(4)),color='red',thickness=RealNumber('1.5'));
      4 geo_hplot2 = parametric_plot3d([Sh[Integer(0)](u = geo_h2[Integer(0)].rhs())(v = geo_h2[Int
eger(1)].rhs()),
      5                                 Sh[Integer(1)](u = geo_h2[Integer(0)].rhs())(v = geo_h2[Integer(1)].rhs()),

/home/justin/bin/Sage/SageMath/local/lib/python2.7/site-packages/sage/plot/plot3d/parametric
_plot3d.pyc in parametric_plot3d(f, urange, vrange, plot_points, boundary_style, **kwds)
   1004         if plot_points == "automatic":
   1005             plot_points = 75
-> 1006         G = _parametric_plot3d_curve(f, urange, plot_points=plot_points, **kwds)
   1007     else:
   1008         if plot_points == "automatic":

/home/justin/bin/Sage/SageMath/local/lib/python2.7/site-packages/sage/plot/plot3d/parametric
_plot3d.pyc in _parametric_plot3d_curve(f, urange, plot_points, **kwds)
   1061     """
   1062     from sage.plot.misc import setup_for_eval_on_grid
-> 1063     g, ranges = setup_for_eval_on_grid(f, [urange], plot_points)
   1064     f_x, f_y, f_z = g
   1065     w = [(f_x(u), f_y(u), f_z(u)) for u in xsrange(*ranges[0], include_endpoint=True)]

/home/justin/bin/Sage/SageMath/local/lib/python2.7/site-packages/sage/plot/misc.pyc in setup_f
or_eval_on_grid(funcs, ranges, plot_points, return_vars)
    148         return fast_float(funcs, *vars,**options), [tuple(range+[range_step]) for range,range_s
tep in zip(ranges, range_steps)], vars
    149     else:
--> 150         return fast_float(funcs, *vars,**options), [tuple(range+[range_step]) for range,range_
step in zip(ranges, range_steps)]
    151
    152

/home/justin/bin/Sage/SageMath/src/sage/ext/fast_eval.pyx in sage.ext.fast_eval.fast_float (/ho
me/justin/bin/Sage/SageMath/src/build/cythonized/sage/ext/fast_eval.c:10920)()
   1381
   1382     if isinstance(f, (tuple, list)):
-> 1383         return tuple([fast_float(x, *vars, expect_one_var=expect_one_var) for x in f])
   1384
   1385     cdef int i

/home/justin/bin/Sage/SageMath/src/sage/ext/fast_eval.pyx in sage.ext.fast_eval.fast_float (/ho
me/justin/bin/Sage/SageMath/src/build/cythonized/sage/ext/fast_eval.c:11141)()
   1396             return f._fast_float_(*vars)
   1397         else:
-> 1398             return fast_callable(f, vars=vars, domain=float, _autocompute_vars_for_backward_c
ompatibility_with_deprecated_fast_float_functionality=True, expect_one_var=expect_one_var)
```

In [ ]: