Computational Geometry Seminar

Wichita State University, Dept. of Mathematics

Frenet Frames and Osculating Circles of Space Curves

Last changed: 11 Sep 2017

Authors: Justin M. Ryan,

Begin by printing the version, formatting the display output, and choosing a 3D-grapher (must be '`jmol`' or '`threejs`' to enable interactive graphs).

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

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

In [2]:
```
%display latex
```

In [3]:
```
viewer3d = 'jmol'
```

Define the space curve $\alpha : \mathbb{R} \to \mathbb{R}^3$ and its graph.

In [4]:
```
var('x y z t');
# define the curve's components
# a = [sin(t)^2+t^2,cos(t)-sin(t),t^5-9];
a = [sin(t),cos(t),t];
# specify the domain (A,B) of the curve segment
A = -4*pi;
B = 4*pi;
# specify the number of Frenet frames you want attached to the curve segment
n=11;
tp = (B-A)/(n-1);
```

In [5]:
```
graph = parametric_plot3d(a,(t,A,B),color='red',xmin=-4*pi, xmax=4*pi, ymin=-4*pi, ymax=4*pi, z
min=-4*pi, zmax=4*pi);
```

In [6]:
```
graph
```

Out[6]:

Calculate the test points along the curve where the Frenet frames will be attached based on the A, B, and n specified above.

In [7]:
```
p = {}
for i in range(n):
    p[i] = [a[0].substitute(t = A+tp*i),a[1].substitute(t = A+tp*i),a[2].substitute(t = A+tp*i)]
```

Compute the tangent vectors to the curve at each test point and plot them.

In [8]:
```
adot = (a[0].diff(t),a[1].diff(t),a[2].diff(t));
L_adot = sqrt(adot[0]^2 + adot[1]^2 + adot[2]^2);
T = (adot[0]/L_adot,adot[1]/L_adot,adot[2]/L_adot);

TT = {}
for i in range(n):
    TT[i] = [T[0].substitute(t=A+tp*i),T[1].substitute(t=A+tp*i),T[2].substitute(t=A+tp*i)];

tar = {}
for i in range(n):
    tar[i] = plot((1/3)*vector(TT[i]), start=vector(p[i]), color='blue', width='1.5');

tangent = sum(tar[i] for i in range(n));
```

In [9]:
```
%display latex
T
show(graph + tangent)
```

Compute the normal vectors at each test point and plot them.

In [10]:
```
Tdot = (T[0].diff(t),T[1].diff(t),T[2].diff(t));
L_Tdot = sqrt(Tdot[0]^2 + Tdot[1]^2 + Tdot[2]^2);
N = (Tdot[0]/L_Tdot,Tdot[1]/L_Tdot,Tdot[1]/L_Tdot);

NN = {}
for i in range(n):
   NN[i] = [N[0].substitute(t=A+tp*i),N[1].substitute(t=A+tp*i),N[2].substitute(t=A+tp*i)];

nar = {}
for i in range(n):
   nar[i] = plot((1/3)*vector(NN[i]), start=vector(p[i]), color='purple', width='1.5');

normal = sum(nar[i] for i in range(n));
```

Compute the binormal vectors at each point and plot them.

In [11]:
```
BB = {}
for i in range(n):
   BB[i] = vector(TT[i]).cross_product(vector(NN[i]));

bar={}
for i in range(n):
   bar[i] = plot((1/3)*vector(BB[i]), start=vector(p[i]), color='green', width='1.5');

binormal = sum(bar[i] for i in range(n));
```

Show all the plots on a single set of axes. The curve is red, tangent vectors are blue, normal vectors are purple, and binormal vectors are green.

In [12]:
```
show(graph+tangent+normal+binormal)
```

Compute the curvature function, $\kappa(t) = \dfrac{||\dot{T}(t)||}{||\dot{\alpha}(t)||}$ and evaluate at all test points.

Note--The helix has constant curvature, so this step is unecessary for this example. However, it is good to have the algorithm in place for when we change the space curve.

In [13]:
```
curv = L_Tdot/L_adot;

k = {}
for i in range(n):
   k[i]=curv.substitute(t=A+tp*i).n()
```

Now plot the osculating circle at a point $p = \alpha(t)$. Recall that $B(t)$ is normal to the osculating plane, the radius of the circle is $\frac{1}{\kappa(t)}$, and the center of the circle lies on the ray determined by $N(t)$.

In [14]:
```
t2 = A + tp*4;
TT3 = vector([T[0].substitute(t=t2).n(),T[1].substitute(t=t2).n(),T[2].substitute(t=t2).n()]);
NN3 = vector([N[0].substitute(t=t2).n(),N[1].substitute(t=t2).n(),N[2].substitute(t=t2).n()]);
BB3 = TT3.cross_product(NN3);
center3 = vector([a[0].substitute(t=t2).n() + N[0].substitute(t=t2).n()/curv.substitute(t=t2).n(),
            a[1].substitute(t=t2).n() + N[1].substitute(t=t2).n()/curv.substitute(t=t2).n(),
            a[2].substitute(t=t2).n() + N[2].substitute(t=t2).n()/curv.substitute(t=t2).n()]);
xx3 = (1/curv.substitute(t=t2).n())*(TT3[0]*cos(t) + NN3[0]*(sin(t) + 1)) + a[0].substitute(t=t2).n();
yy3 = (1/curv.substitute(t=t2).n())*(TT3[1]*cos(t) + NN3[1]*(sin(t) + 1)) + a[1].substitute(t=t2).n();
zz3 = (1/curv.substitute(t=t2).n())*(TT3[2]*cos(t) + NN3[2]*(sin(t) + 1)) + a[2].substitute(t=t2).n();
osc_circ = parametric_plot3d([xx3,yy3,zz3],(t,0,2*pi),color='black');
```

In [15]:
```
show(graph + osc_circ)
```

Now that we have the algorithm for a single point, use a `for` loop to store them at every test point, as above.

In [16]:
```
xx= {}
yy={}
zz={}

for i in range(n):
   xx[i] = (1/curv.substitute(t = A + tp*i).n())*(T[0].substitute(t = A + tp*i).n()*cos(t)
                                  + N[0].substitute(t = A + tp*i).n()*(sin(t) + 1)) + a[0].substitute(t = A +
tp*i).n();
   yy[i] =(1/curv.substitute(t = A + tp*i).n())*(T[1].substitute(t = A + tp*i).n()*cos(t)
                                  + N[1].substitute(t = A + tp*i).n()*(sin(t) + 1)) + a[1].substitute(t = A +
tp*i).n();
   zz[i] = (1/curv.substitute(t = A + tp*i).n())*(T[2].substitute(t = A + tp*i).n()*cos(t)
                                  + N[2].substitute(t = A + tp*i).n()*(sin(t) + 1)) + a[2].substitute(t = A +
tp*i).n();

osc = {}

for i in range(n):
   osc[i] = parametric_plot3d([xx[i],yy[i],zz[i]],(t,0,2*pi),color='black');

circles = sum(osc[i] for i in range(n));
```

Finally, plot it all on the same set of axes.

In [17]:
```
show(graph + tangent + normal + binormal + circles)
```

Whoa. Or, maybe just plot some of the circles.

In [18]:
```
show(graph + tangent + normal + binormal + osc[4] + osc[5] + osc[6],aspect_ratio=1)
```

We could also use the `@interact` feature to cycle through one frame/circle at a time.

In [19]:
```
%display plain
@interact
def _(index=(((n)/2).floor(),(0,n-1))):
  show(graph + tar[index] + nar[index] + bar[index] + osc[index])
```