Sage_Talk_5_29_08

VISUALIZING COMPLEX ANALYTIC FUNCTIONS

Any good things in this talk should be credited to William, Tom, and Robert Miller. There is lots of bad stuff that I added. And there were many false starts. Thanks for asking me to talk. This way I get to ask the questions with an expert audience.

Don Marshall has a terrific conformal mapping program, Zipper: http://www.math.washington.edu/~marshall/zipper.html. You might want to ask him if you can add it to Sage. Some more references:

Conformal mapping has gotten sophisticated. New methods for multiply-connected domains: T.K. DeLillo, T.A. Driscoll, A.R. Elcrat, and J.A. Pfaltzgraff, *Computation of multiply connected Schwarz–Christoffel maps for exterior domains* , Comput. Methods and Function Theory, 6:2 (2006), 301–315

And more numerical methods: T.A. Driscoll and L.N. Trefethen, *Schwarz–Christoffel Mapping* , Cambridge University Press, London and New York, 2002.

The derivative of a non-constant complex analytic function $f(z)$ has isolated zeros. That means that f is conformal (preserves angles of intersecting curves) at most points. Vectors in the tangent space T_p at a point where $f'(p) \neq 0$ are multiplied by $f'(p)$ when tangent vectors are represented by complex numbers. So all tangent vectors are rotated by $Arg(f'(p))$ and stretched by the same amount, $|f'(p)|$.(An alternate definition: small circles are mapped to small circles (not small ellipses). The stretch is the same in all directions.) However at zeros of the derivative of order m, curves that intersect at angle θ map to curves that intersect at angle $(m+1)\theta$. How can this happen? Is this even a good statement. You will find this statement (and a proof) in many texts.

Shouldn't the derivative act continuously and not jump as in quantum mechanics?

The confusion is caused by in one case looking at the map of tangent spaces and in the other case looking at a map of *rays*. In any case the statement should be that rays from the critical point that make *small* angle θ with each other map to (approximate) rays that make angle $(m+1)\theta$ with each other. It is best illustrated by looking at maps of the form $z \to z^n$. In the first figure red and green *lines* making a small angle map by z^6 to red and green *rays*, but red and green rays map to red and green rays. Can you guess what happens if the exponent is odd?

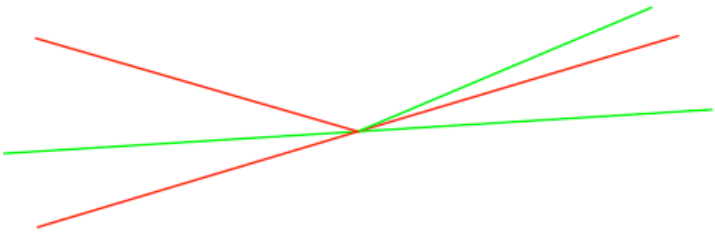
```
var('t')
```

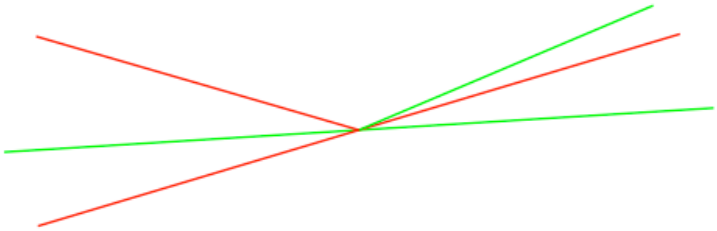
```
t
```

```
@interact
def _(a=(0.15,1.0)):
   j1(t)= t*exp(.1*I)
   j2(t)= t*exp(a*I)
   print "a = ", a
   pj1=parametric_plot((j1.real(), j1.imag()),-1,1, color="green")
   pj2=parametric_plot((j2.real(), j2.imag()),-1,1, color="red")
   sj1(t)=(j1(t))^6
   sj2(t)=(j2(t))^6
   psj1=parametric_plot((sj1.real(), sj1.imag()),-1,1,
color="green")
   psj2=parametric_plot((sj2.real(), sj2.imag()),-1,1, color="red")
   ps=pj1+pj2+psj1+psj2
   ps.axes(False)
   show(ps)
```

a

```
a =  0.450505050505051
```



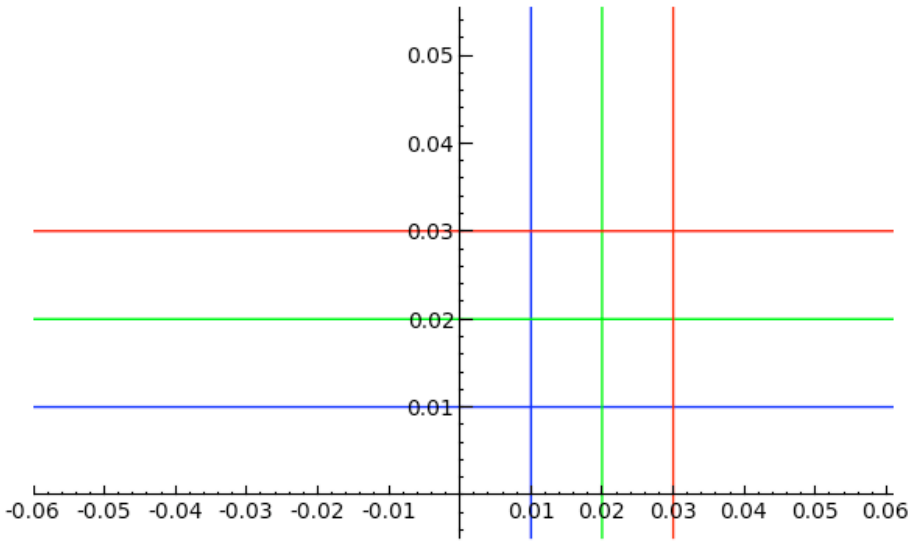


```
l1(t)= t+.01*I ; l2(t)=.01+t*I; l3(t)= t+.02*I;
l4(t) = .02+t*I; l5(t)= t+.03*I; l6(t) = .03+t*I
```

A grid of lines intersecting at points in the first quadrant. The intersection point approaches $(0, 0)$.

```
m1=parametric_plot((l1.real(), l1.imag()), -1,1)
m2=parametric_plot((l2.real(), l2.imag()), -1,1)
m3=parametric_plot((l3.real(), l3.imag()), -1,1, color="green")
```

```
m4=parametric_plot((l4.real(), l4.imag()), -1,1, color="green")
m5=parametric_plot((l5.real(), l5.imag()), -1,1, color="red")
m6=parametric_plot((l6.real(), l6.imag()), -1,1, color="red")
show(m1+m2+m3+m4+m5+m6, xmin = -.05, xmax = .05, ymin = 0, ymax =
.05)
```

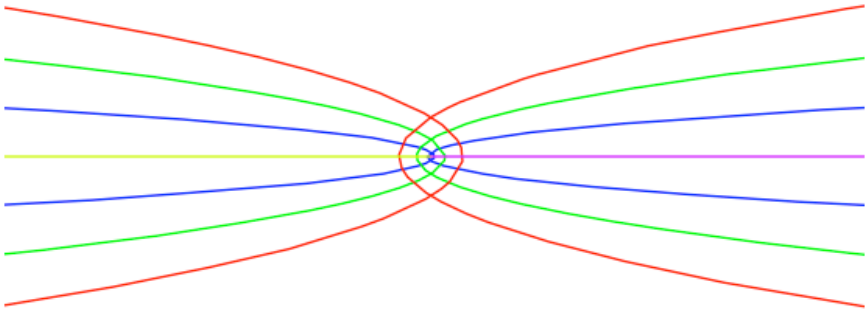



```
sq1(t) = (t+.01*I)^2
sq2(t) = (.01+t*I)^2
sq3(t) = (t+.02*I)^2
sq4(t) = (.02+t*I)^2
sq5(t) = (t+.03*I)^2
sq6(t) = (.03+t*I)^2
sq7(t) = (t*I)^2
sq8(t)=(t*CDF(1,0))^2
```

The images under the function $f : z \to z^2$. Notice the straight lines are mapped to curves that bend sharply in the neighborhood of $(0, 0)$. The entire real axis maps to the positive real axis and the entire imaginary axis maps to the negative real axis (each image is a ray). So f maps these smooth curves through the origin to curves which are not smooth (each ray has a singular point at the origin). So it is incorrect to say that the angle between the image curves is the double of the angle between the original curves. It is true that the positive real and imaginary axis map to rays which make an angle of 180 degrees with each other, so the angle between these rays is doubled, but as a map of tangent vectors, $f'(0) = 0$ is the 0 map. The purple and yellow curves are images of (axis) lines

through the origin and are rays and hence not smooth curves.

```
p1=parametric_plot((sq1.real(), sq1.imag()),-1,1)
p2=parametric_plot((sq2.real(), sq2.imag()),-1,1)
p3=parametric_plot((sq3.real(), sq3.imag()),-1,1, color="green")
p4=parametric_plot((sq4.real(), sq4.imag()),-1,1, color="green")
p5=parametric_plot((sq5.real(), sq5.imag()),-1,1, color="red")
p6=parametric_plot((sq6.real(), sq6.imag()),-1,1, color="red")
p7=parametric_plot((sq7.real(), sq7.imag()),-1,1, rgbcolor=hue(.2))
p8=parametric_plot((sq8.real(), sq8.imag()),-1,1, rgbcolor=hue(.8))
p=p1+p2+p3+p4+p5+p6+p7+p8
p.axes(False)
show(p, xmin = -.01, xmax = .01, ymin = -.01, ymax = .01)
```



Next I'll look at the image of the complex exponential on rays through the origin with arguments of the form $-pi/2 + j\pi/6$, where $j = 0, \ldots, 5$. These are somewhat familiar logarithmic spirals that wind in around 0 and out to ∞. I'll superimpose images of circles of radii $r = (j+1)/4, j = 0, \ldots, 5$ These circles are of moderate size so the modulus of $e^{re^{it}}$ varies from e^r to e^{-r} and so the image can be seem on (sort of) the same scale. However it is easy to not notice that the argument of $e^{re^{it}}$ is $r \sin t$ and as t varies from 0 to 2π it varies between $-r$ and r. If $r < \pi$ the image curve comes close to the origin, but never winds around it (winding number 0). This is not so obvious from the graph unless we magnify near the origin. These closed curves are orthogonal to the spirals.

```
v=[]
for j in range(6):
   f(t)=exp(t*exp((-pi/2 +j*pi/6)*I))
   v.append(f)
```

```
x=[]
for j in range(6):
  p=parametric_plot((v[j].real(), v[j].imag()), -4,4, color="blue")
  x.append(p)
```

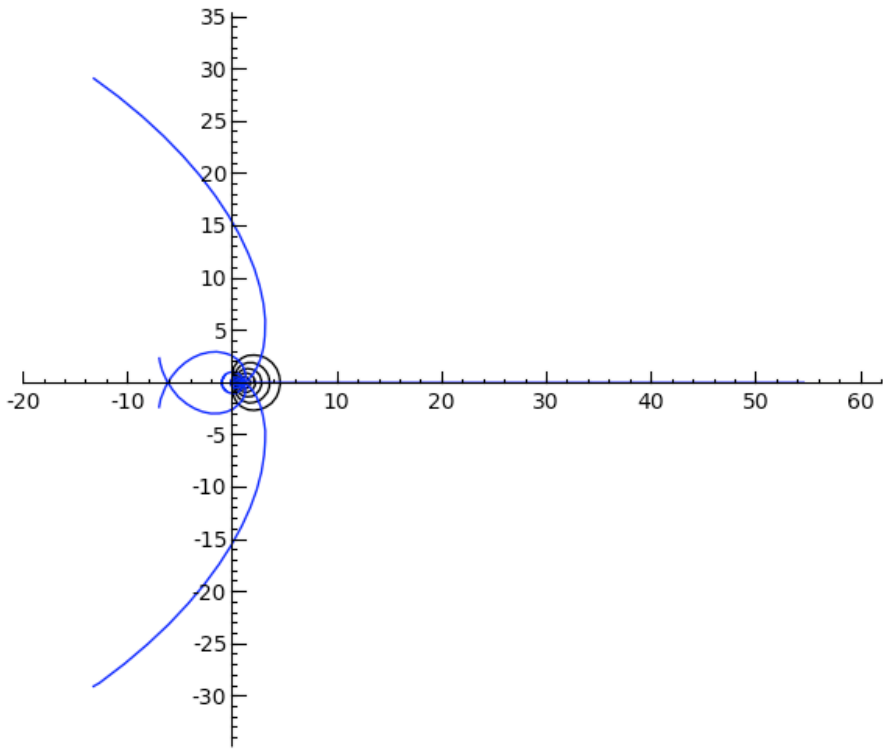
```
xx=x[0]+x[1]+x[2]+x[3]+x[4]+x[5]
```

```
w=[]
for j in range(6):
   g(t)=exp(((j+1)/4)*exp(t*I))
   w.append(g)
```

```
y=[]
for j in range(6):
 q=parametric_plot((w[j].real(), w[j].imag()), -pi,pi,
color="black")
 y.append(q)
```

```
yy=y[0]+y[1]+y[2]+y[3]+y[4]+y[5]
```

```
show(yy+xx, aspect_ratio=1)
```



What happens to circles of larger radii? Do the images wind around 0? At first glance it appears that they do. But this can't happen. The argument principle says that if they did, then 0 would be an assumed value of e^z. But this winding number can be computed on a circle centered at the origin of radius r: Let n be the number of zeros of e^z inside the circle of radius r.

$$n = \frac{1}{2\pi i}\int_{|z|=r}\frac{f'(z)}{f(z)}dz = \frac{1}{2\pi i}\int_{|z|=r}\frac{e^z}{e^z}dz = 0$$

(Something is wrong with the integral symbols.)

So what is going on with the curves? I'll put the microscope on some of the plots to take a closer look.

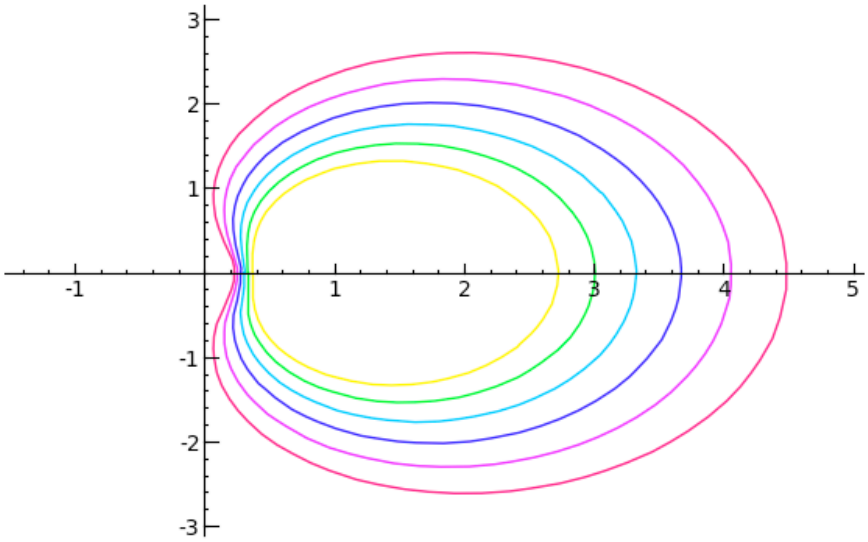
```
hz=[]
for j in range(6):
   hh(t)=exp((.1*j+1)*exp(t*I))
   hz.append(hh)
```

```
hp=[]
for j in range(6):
 q=parametric_plot((hz[j].real(), hz[j].imag()),
```

```
-pi,pi,rgbcolor=hue(sin(.2*j+1/6)))
 hp.append(q)
```

```
hzz=hp[0]
for j in [1..5]:
   hzz=hzz+hp[j]
```

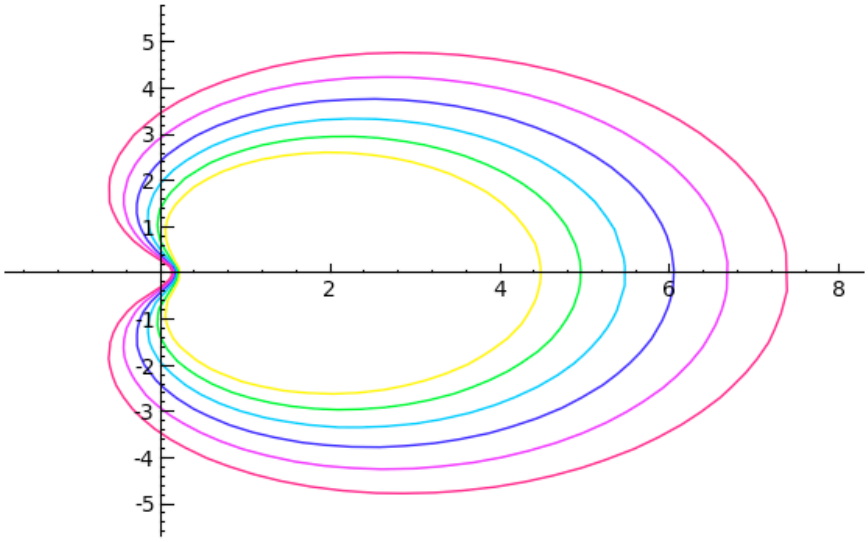
```
show(hzz)
```




```
hz=[]
for j in range(6):
   hh(t)=exp((.1*j+1.5)*exp(t*I))
   hz.append(hh)
```

```
hp=[]
for j in range(6):
 q=parametric_plot((hz[j].real(), hz[j].imag()),
-pi,pi,rgbcolor=hue(sin(.2*j+1/6)))
 hp.append(q)
```

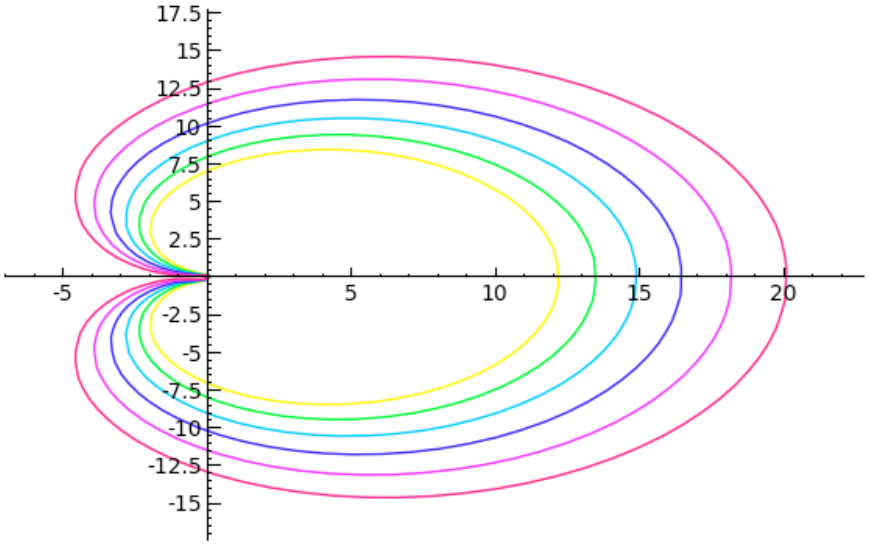
```
hzz=hp[0]
for j in [1..5]:
   hzz=hzz+hp[j]
show(hzz)
```



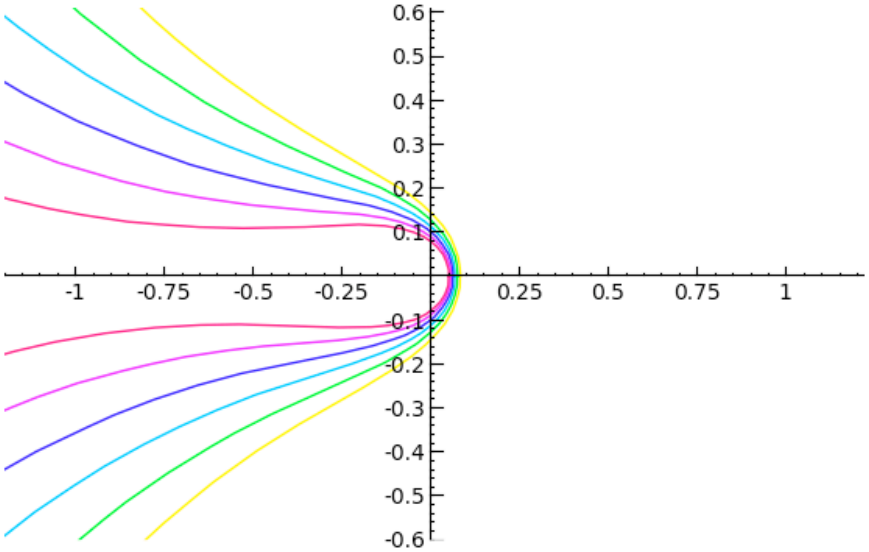

```
hz=[]
for j in range(6):
   hh(t)=exp((.1*j+2.5)*exp(t*I))
   hz.append(hh)
```

```
hp=[]
for j in range(6):
 q=parametric_plot((hz[j].real(), hz[j].imag()),
-pi,pi,rgbcolor=hue(sin(.2*j+1/6)))
 hp.append(q)
```

```
hzz=hp[0]
for j in [1..5]:
   hzz=hzz+hp[j]
show(hzz)
```




```
show(hzz, xmin=-1, xmax=1, ymin=-.5, ymax=.5)
```

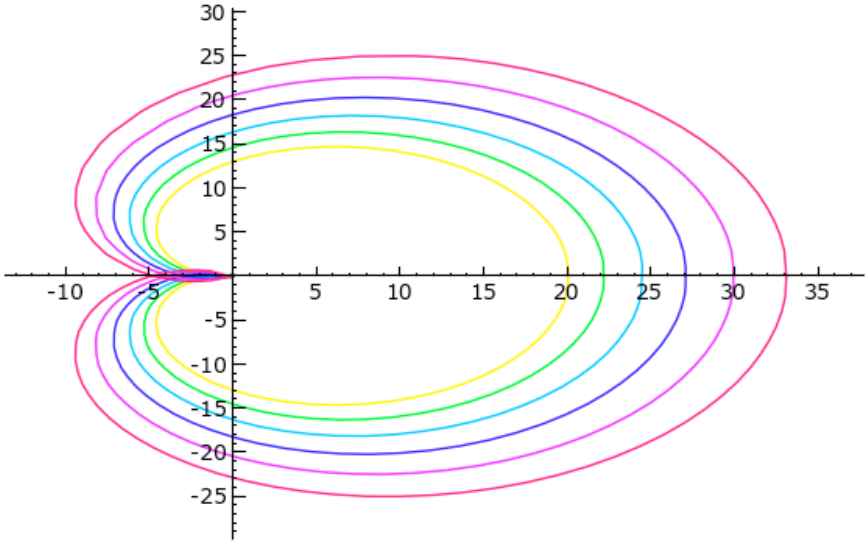



```
hz=[]
for j in range(6):
   hh(t)=exp((.1*j+3)*exp(t*I))
   hz.append(hh)
```

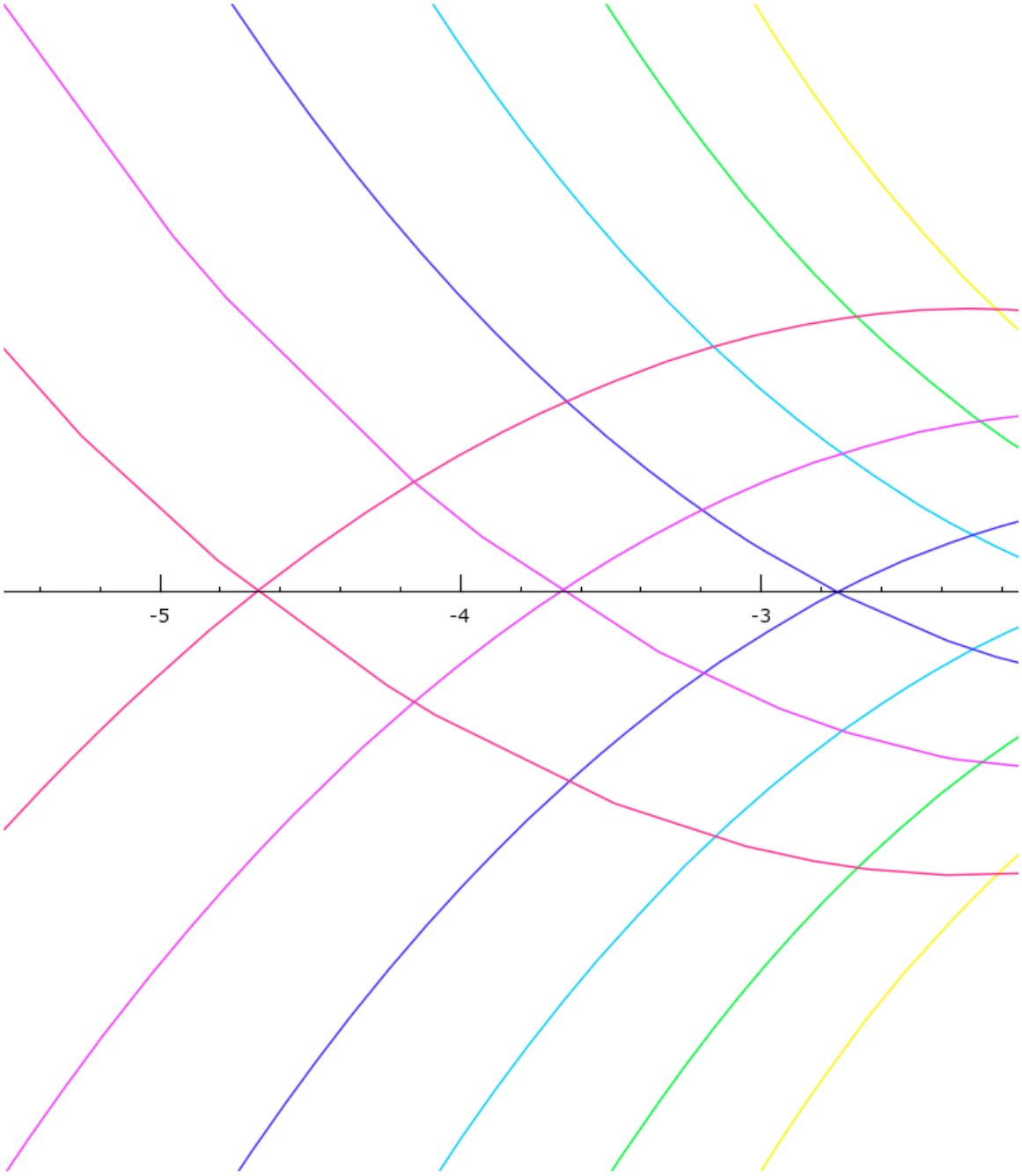
```
hp=[]
for j in range(6):
 q=parametric_plot((hz[j].real(), hz[j].imag()),
-pi,pi,rgbcolor=hue(sin(.2*j+1/6)))
 hp.append(q)
```

```
hzz=hp[0]
for j in [1..5]:
```

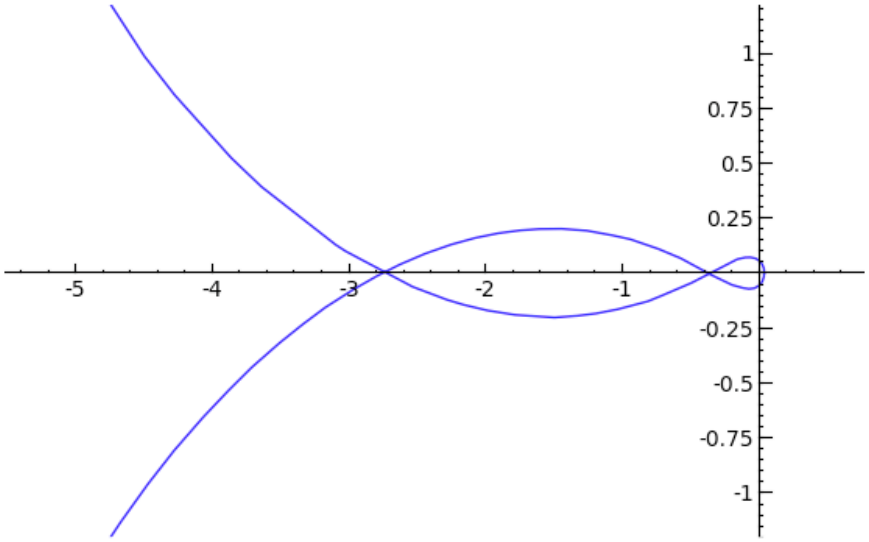
```
    hzz=hzz+hp[j]
show(hzz)
```




```
show(hzz, xmin=-5, xmax=.2, ymin=-1, ymax=1, figsize=15)
```

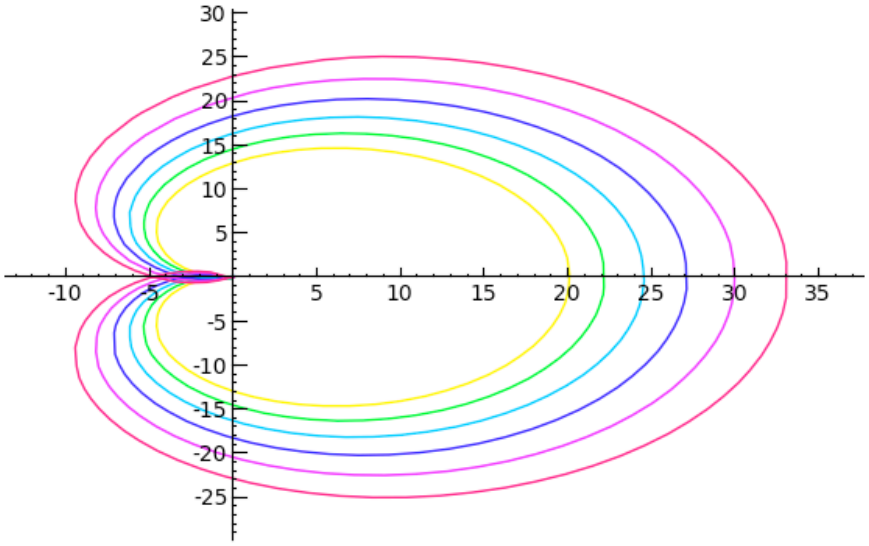



```
show(hp[3], xmin=-5, xmax=.2, ymin=-1, ymax=1)
```



Play around with the radius. On a large scale it looks like there is a singularity at the origin. But this can't be. The curves wind tightly around the origin.

```
v=[]
for j in range(6):
   h(t)=exp((.1*j+3)*exp(t*I))
   v.append(h)
vv=[]
for j in range(6):
 q=parametric_plot((v[j].real(), v[j].imag()),
-pi,pi,rgbcolor=hue(sin(.2*j+1/6)))
 vv.append(q)
vvv= sum(vv[j] for j in range(6))
show(vvv)
```

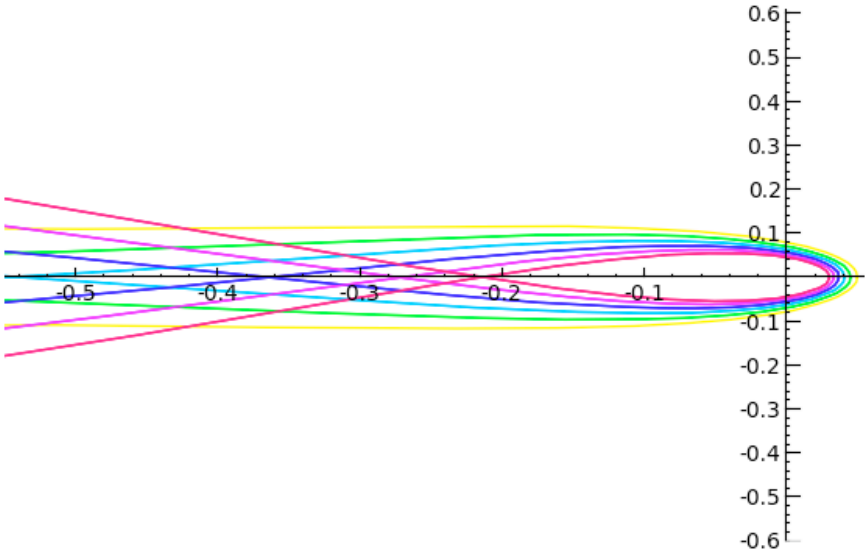



```
z=[]
for j in range(6):
   h(t)=exp((.1*j+3)*exp(t*I))
   z.append(h)
```

```
zz=[]
for j in range(6):
 q=parametric_plot((z[j].real(), z[j].imag()),
-pi,pi,rgbcolor=hue(sin(.2*j+1/6)))
 zz.append(q)
```

```
zzz=zz[0]
for j in [1..5]:
  zzz=zzz+zz[j]
```

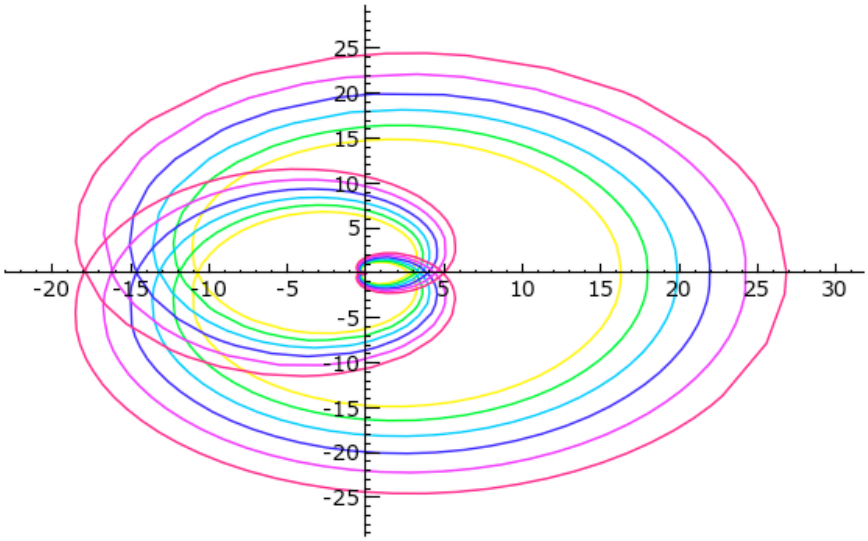
```
show(zzz, xmin=-.5, xmax=0, ymin=-.5, ymax=.5)
zzz.save("plot.eps", xmin=-5, xmax=0, ymin=-2, ymax=2)
```



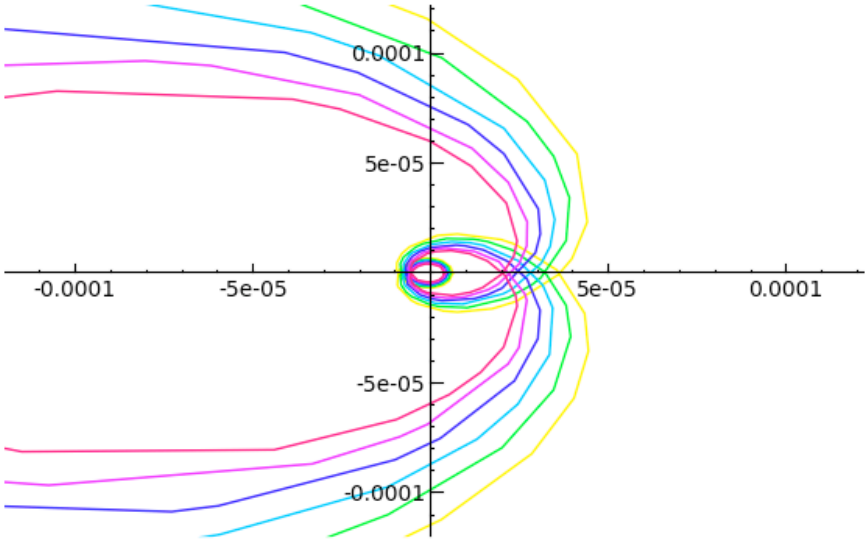
plot.eps

Make the radius large. Scale the image. How do I add more plot points?

```
v=[]
for j in range(6):
   h(t)=.0001*exp((.1*j+12)*exp(t*I))
   v.append(h)
vv=[]
for j in range(6):
 q=parametric_plot((v[j].real(), v[j].imag()),
-pi,pi,rgbcolor=hue(sin(.2*j+1/6)),plot_points=500)
 vv.append(q)
vvv= sum(vv[j] for j in range(6))
show(vvv)
```



```
v=[]
for j in range(6):
   h(t)=exp((.1*j+12)*exp(t*I))
   v.append(h)
vv=[]
for j in range(6):
 q=parametric_plot((v[j].real(), v[j].imag()),
-pi,pi,rgbcolor=hue(sin(.2*j+1/6)))
 vv.append(q)
vvv= sum(vv[j] for j in range(6))
show(vvv, xmin=-.0001, xmax=.0001, ymin=-.0001, ymax=.0001)
```



This one is really knotty. Figures are in reverse order. The bulges push around and enter the right half-plane. Then it would become clearer that the bug doesn't encircle the origin. It goes around and then "reverses" direction. No wonder the Jordan curve theorem is tricky to prove. How can you tell when a point is *inside* a simple closed curve? (One way is to compute the winding number which is an integer and perhaps can be computed accurately.)

Is there a way to make a movie with a bug running around the curves?

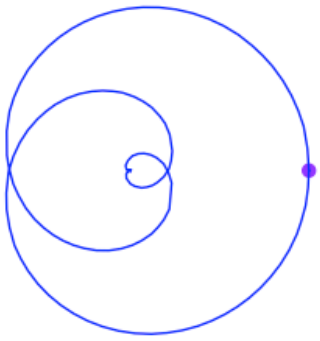
```
k(t)=.00001*exp(4*pi*1.05*exp(t*I))
```

```
kk=parametric_plot((k.real(),k.imag()),0,2*pi)
```

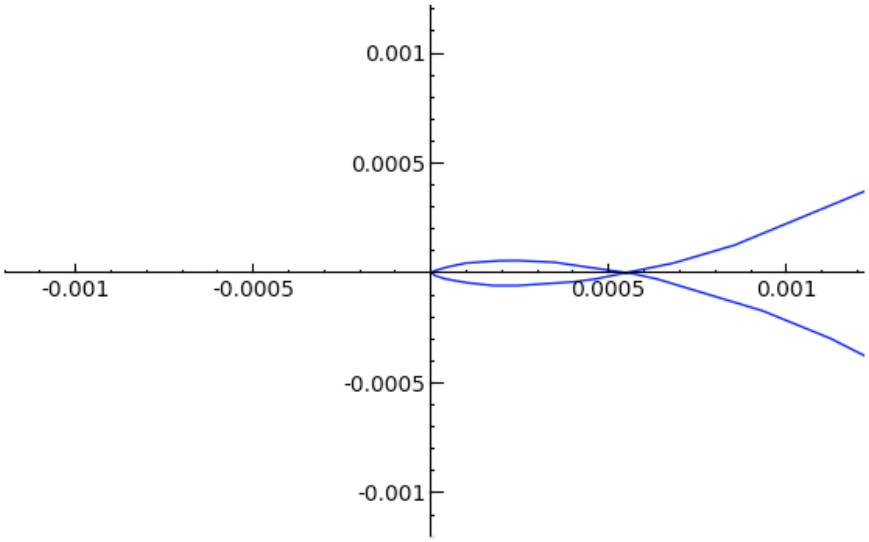
```
@interact
def _(t=(0,2*math.pi)):
   w = k(t)
```

```
    x,y = w.real(), w.imag()
    eps = 2*abs(x)
    show(kk + point((k(t).real(), k(t).imag()),
          rgbcolor='purple',pointsize=50), aspect_ratio=1,
              xmin=x-eps,xmax=x+eps,ymin=y-eps,ymax=y+eps ,
axes=False)
```

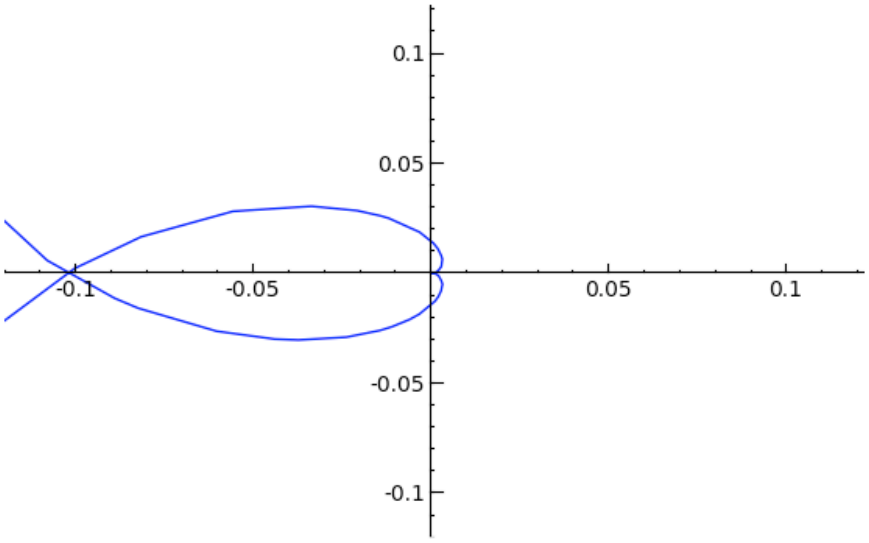
t



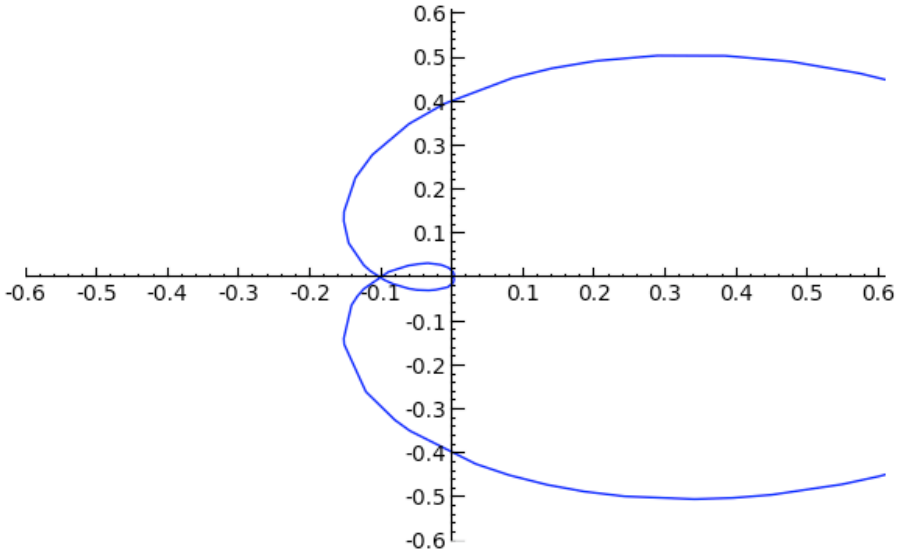
```
kkk=parametric_plot((k.real(),k.imag()),.56,  5.55)
show(kkk, xmin=-.001, xmax=.001, ymin=-.001, ymax=.001)
```




```
kkk=parametric_plot((k.real(),k.imag()),-.6*pi, .6*pi)
show(kkk, xmin=-.1, xmax=.1, ymin=-.1, ymax=.1)
```

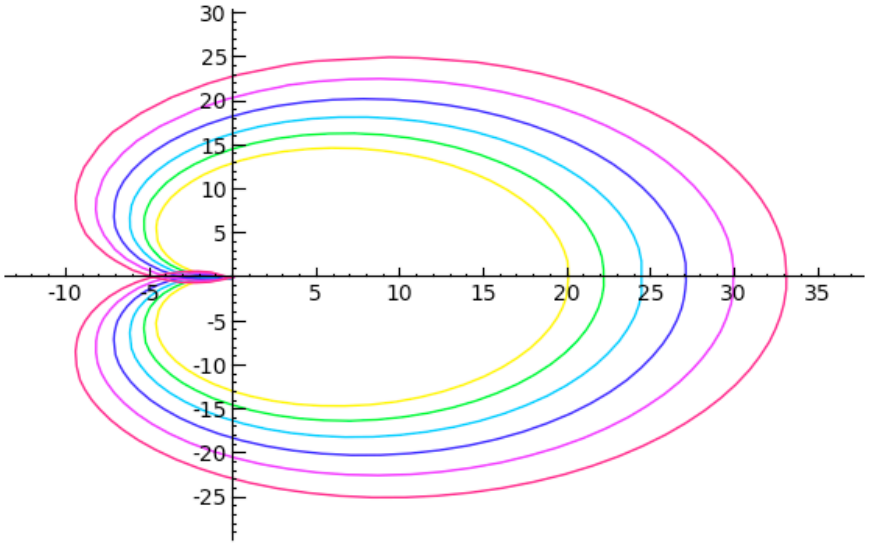



```
kkk=parametric_plot((k.real(),k.imag()),-.6*pi, .6*pi)
show(kkk, xmin=-.5, xmax=.5, ymin=-.5, ymax=.5)
```



Duplication. Who knew the exponential is so complicated?

```
v=[]
for j in range(6):
   h(t)=exp((.1*j+3)*exp(t*I))
   v.append(h)
vv=[]
for j in range(6):
 q=parametric_plot((v[j].real(), v[j].imag()),
-pi,pi,rgbcolor=hue(sin(.2*j+1/6)))
 vv.append(q)
vvv= sum(vv[j] for j in range(6))
show(vvv)
```


30
25
20
15
10
5
-10
-5
5
10
15
20
25
30
35
-5
-10
-15
-20
-25