Visualization of Number Theory with SAGE

Alex Clemesha

University of Washington

May 25 2006

Outline

1. **Implementation**
   Wrapping matplotlib classes
2. **Basic Functions**
   Functions provided to users
3. **Plotting**
   Making image files from SAGE functions
4. **Visualizing Number Theory (the fun part)**
   Using SAGE mathematical and graphics functions together

- **Desire:**
  We want **SAGE** to have excellent graphics capabilities
- **Solution:**
  Gnuplot license says: " Permission to modify the software is granted, but not the right to distribute the complete modified source code."

  PyX, still considering, 3D capabilities look good, but no pngs, svg, but *not* ruled out.

  Use **matplotlib** by John Hunter (http://matplotlib.sourceforge.net)!

  ...but **matplotlib** provides users a very "Matlab-like" interace, because after all thats what it was designed to do. So instead we wrap matplotlib's classes with our own more "Mathematica-like" interface.

- **class that handles all the data of a graphic**

  `from matplotlib.figure import Figure`

- **classes that handle the generation of the image files**

  for png files:

  `FigureCanvasAgg`

  for ps or eps files:

  `FigureCanvasPS`

  for svg files (svg is an XML graphics format):

  `FigureCanvasSVG`

Basics functions

- plotting

```
plot(f, xmin, xmin, **options)
parametric_plot((fx, fy), xmin, xmax, **options)
list_plot(L, **options)
```

- Graphics objects

```
circle((x, y), radius)
disk((x, y), radius, theta1, theta2)
line(xydata)
point((x, y))
polygon(xydata)
text(string, (x, y))
```

Making the image files

- Once you have Graphics object g you can:

```
g.show(**options)
g.save(**options)
```

- If you have several Graphics objects g1,...,gn you can:

```
ga = graphics_array([[g1, g2],[g3, g4]])
ga.show(**options)
ga.save(**options)
```

User defined functions

Two (equivalent) ways of creating user defined functions:

- Regular functions:

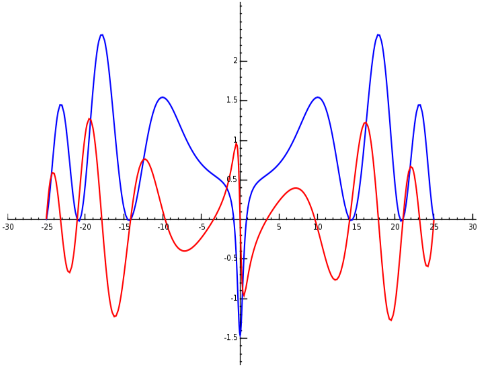
```
def zrf(t):
    return zeta(1/2 + I*t).real()

def zif(t):
    return zeta(1/2 + I*t).imag()
```

- Lambda functions:

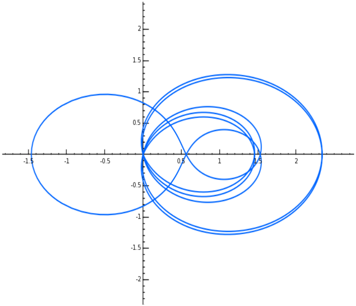
```
zrl = lambda t: zeta(1/2 + I*t).real()
zil = lambda t: zeta(1/2 + I*t).imag()
```

```
p1 = plot(zrf,-25,25,rgbcolor=(0,0,1))
p2 = plot(zif,-25,25,rgbcolor=(1,0,0))
(p1 + p2).save('zeta.png',figsize=[4,4])
```



It is conjectured that all values $s = \sigma + it$ such that $\zeta(s) = 0$, $\sigma > 0$, have the form $s = 1/2 + it$. Here we look at the real part of $\zeta(s)$ versus the imaginary part.

```
p3 = parametric_plot((zrl, zil),-25,25, \
     rgbcolor=hue(0.6),plot_points=1000)
p3.save('zetap.png',ymin=-2,ymax=2,figsize=[3,3])
```



Proposition: Let p be a prime number congruent to 1 mod 4. There exists a right triangle with integer sides such that the length of the hypotenuse is p (e.g., 5, 13, 17). In $\mathbb{Z}[i] = \{a + bi | a, b \in \mathbb{Z}\}$ the numbers p lose their irreducibilty, so we can factor them there. For example: $13^2 = (3 + 2i)^2(3 - 2i)^2 = 5^2 + 12^2$

```
#prime number congruent to 1 mod 4:
L = [p for p in primes(30) if (p-1)%4 == 0]; L
[5,13,17,29]
```

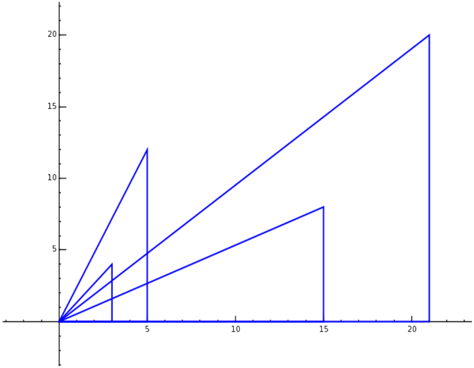
```
#construct the Number Field
K.<I> = NumberField(x^2 + 1)
K.factor_integer(13)[0][0].gens_reduced()[0]
3*I - 2
```

```
gg = Graphics() #empty Graphics object
gl = [] #empty list

for p in L:
    z = K.factor_integer(p)[1][0].gens_reduced()[0]
    zz = z^2
    a,b = abs(zz[0]),abs(zz[1])
    lv = [[0, 0], [a, 0], [a, b], [0, 0]]
    l = line(lv, rgbcolor=(0,0,1))
    sv = (a, b, sqrt(a^2 + b^2))
    s = "$(%s,\ %s,\ %s)$"%sv
    t = text(s, (2*a/3, b/4), fontsize=8)
    gg += l
    gl.append((l + t))
```

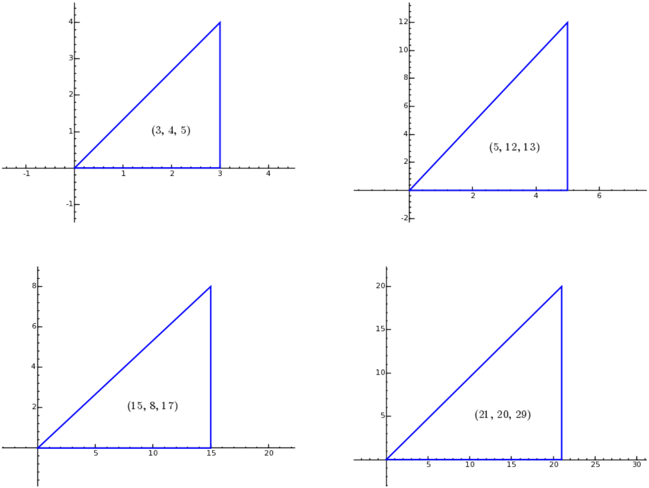
Here is the result of connecting the points we found from the above code which factors prime numbers in $\mathbb{Z}[i]$.

```
gg.save('triples1.png',figsize=[6,4])
```



Here is a graphics_array of the above triangles.

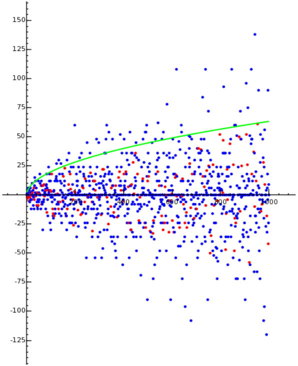
```
graphics_array([gl[0:2],gl[2:4]]).save('triples2.png')
```

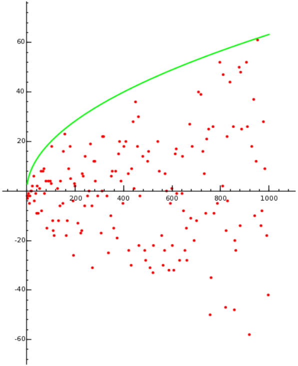


Here we take the first 1000 coefficients of the $q-$expansion of the modular form corresponding to the elliptic curve $y^2 + y = x^3 + x^2 - 2x$.

```
E = EllipticCurve("37a")
ans = E.anlist(1000)
g,h = Graphics(),Graphics()
m = abs(max(ans))
for i,an in zip(range(len(ans)),ans):
    c = (0,0,1) #blue
    if is_prime(i):
        c = (1,0,0) #if prime color point red
        h += point((i,an), pointsize=2, rgbcolor=c)
    g += point((i,an), pointsize=2, rgbcolor=c)
g += plot(lambda x: 2*sqrt(x), 2, n, rgbcolor=(0,1,0))
h += plot(lambda x: 2*sqrt(x), 2, n, rgbcolor=(0,1,0))
g.save("ec37an.eps",figsize=[3,3])
h.save("ec37ap.eps",figsize=[3,3])
```

It is a theorem by Hasse that $|a_p| \leq 2\sqrt{p}$ for all primes p. Prime points are red, non-prime points are blue and the line $2\sqrt{p}$ is green.

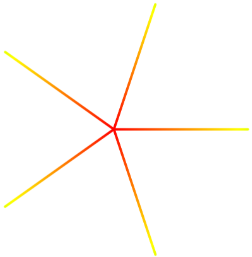


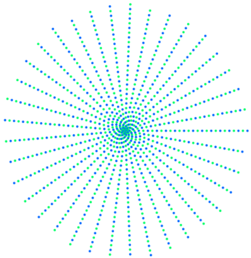


Here is some SAGE code that draws spirals for a given constant c.

```
c = 37/41
g = Graphics()
for k in range(1000):
    xr = k*cos(2*pi*c*k)
    yr = k*sin(2*pi*c*k)
    g += point((xr, yr), rgbcolor=hue(0.4+0.2*(k%2)))
g.save('spiral.png', figsize=[6,6], draw_axis=False)
```

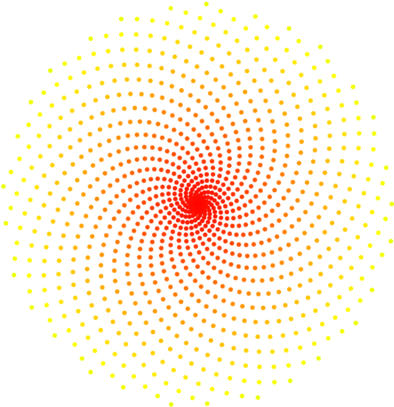
If you have any $c \in \mathbb{Q}$ you will always eventually see 'arms' appear, i.e., the point $(cos(2\pi ck), sin(2\pi ck))$ will repeat as k runs through the integers. For the below examples $k = 0, ..., 2000$.





first with $c = 1/5$ then with $c = 37/41$.

Above code with $c = \sqrt{2}$. Note that $\sqrt{2}$ has the continued fraction $1 + \frac{1}{2+\frac{1}{2+\frac{1}{2+..}}}$.



c = golden ratio. Note that the golden ratio has continued fraction. $1 + \frac{1}{1+\frac{1}{1+\frac{1}{1+..}}}$.