Lotka-Volterra Equation

The Lotka-Volterra equation is a simple model of prey and predator populations. Let x be the prey population (say, the number of rabbits, scaled somehow) and y is the predator population (say, foxes). The frequency of a fox meeting a rabbit is proportional to xy. For the rabbits, we have this:
$\dot{x} = \alpha x - \beta xy$
which means that the rabbits would exponentially reproduce as $e^{\alpha t}$ but the casualties are proportional to frequency of meeting a fox. For the foxes, the equation goes similarly:
$\dot{y} = -\gamma y + \delta xy$
and that means that foxes are busy converting consumed rabbits into new foxes, but would die out exponentially if they cannot eat.

Here's a straightforward implementation:

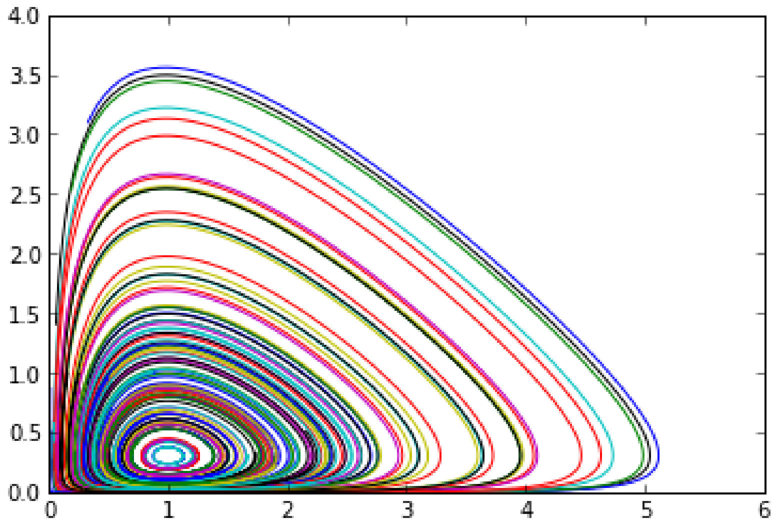
In [1]:
```
def lotka_volterra(alpha, beta, gamma, delta, end_time, x0, y0,
    t = arange(0, end_time, time_step)
    x = zeros(len(t))
    y = zeros(len(t))

    x[0] = x0
    y[0] = y0

    for i in range(1, len(t)):
        dx = x[i-1] * (alpha - beta * y[i-1])
        dy = y[i-1] * (delta * x[i-1] - gamma)
        x[i] = x[i-1] + time_step * dx  # the simulation step
        y[i] = y[i-1] + time_step * dy  # the simulation step

    return x, y
```

But let's stop and think how many parameters we're actually interested in. We don't really care if the units of measurement change, so let's start with time. If we change variable $t' = t/k$, we get
$dx/dt' = dx/d(t/k) = kdx/dt = k\dot{x} = k\alpha x - k\beta xy$
and same with y. This means that if all parameters ($\alpha, \beta, \gamma, \delta$) are multiplied by the same number, the trajectory doesn't change.

Then, we're also free to choose the scale for x and y. Observe that rescaling x only changes δ and rescaling y only changes γ, therefore, we can set $\gamma = \delta = 1$ and only the scale for x and y changes.

So we're actually left with one parameter, α; all the rest may be set to 1 without losing the behavior.

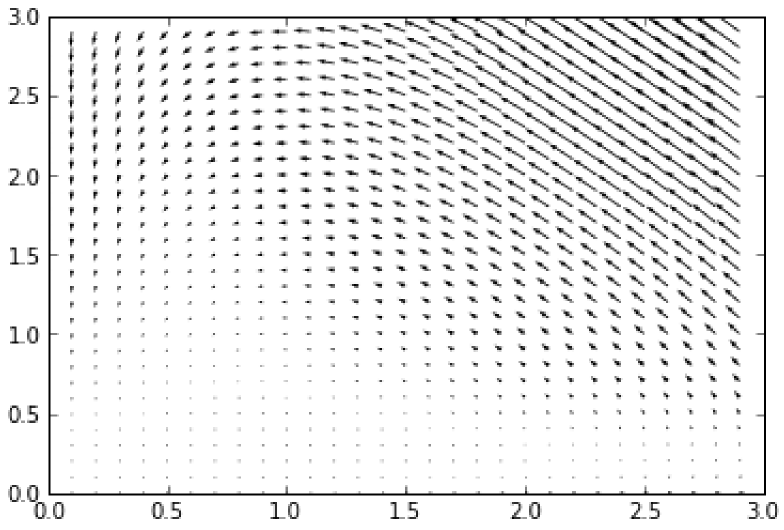
In [2]:
```
alpha = 0.3
for i in range(100):
    x, y = lotka_volterra(alpha, 1, 1, 1,
                          20, # step
                          random.random(), random.random())
    plot(x, y)
```



In [4]:
```
X,Y=mgrid[0:3:0.1, 0:3:0.1]
```

In [5]:
```
alpha = 0.3
quiver(X, Y, X*(alpha - Y), Y * (X - 1))
```

Out[5]:
```
<matplotlib.quiver.Quiver at 0x2cae410>
```



In [ ]: