Building Differential Equation Models

The Epidemic Curve

Sir Ronald Ross observed infectious diseases come in three forms:

- endemic - small number of cases, fluctuating, only slow increase/decrease
- outbreak - constantly present but flaring up in epidemic outbreaks
- epidemic - intense outbreaks followed by disappearance

What's the reason?

SIR Models

$$S \to I \to R$$

Susceptible:

$$\dot{S} = -\beta SI$$

Infected:

$$\dot{I} = \beta SI - \gamma I$$

Removed (resistant or dead):

$$\dot{R} = \gamma I$$

Total population: N

Initial population: $S_0 \approx N$

Interpretation

- each infected individual transmits the pathogen to b others per unit time
- number of new cases per unit time $bI(S/N) = \beta SI$ where $\beta = b/N$
- γ is the rate of recovery, or equivalently, the mean duration of illness (Poisson process)

Rescaling the model

Express values as fraction of total population: $X = S/N$, $Y = I/N$, $Z = R/N$.

$$b = \beta N$$

$$\dot{X} = -bYZ$$

$$\dot{Y} = bXY - \gamma Y$$

$$\dot{Z} = \gamma Y$$

We have two parameters now: b and γ .

Rescaling in time

We can eliminate the γ parameter by rescaling in time as well:

- γ is mean duration of infection
- express time as a multiple of this $\tau = t/\gamma$
- this reduces the equations to a single parameter $\rho = \beta N/\gamma$
- ρ is the *basic reproductive rate* of the disease

$$\dot{X} = -\rho XY$$

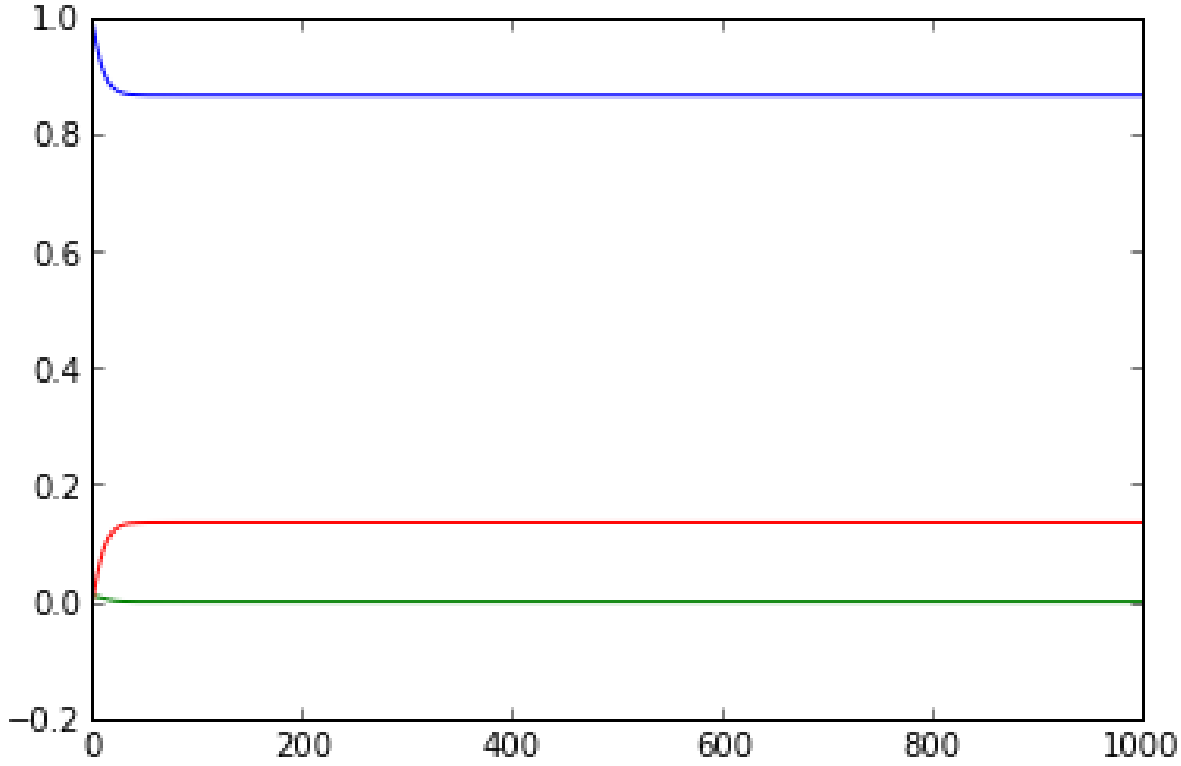
$$\dot{Y} = \rho XY - Y$$

$$\dot{Z} = Y$$

```
def F(rho):
    def f(v,_): x,y,z = tuple(v); return (-rho*x*y,rho*x*y-y,y)
    return f
```

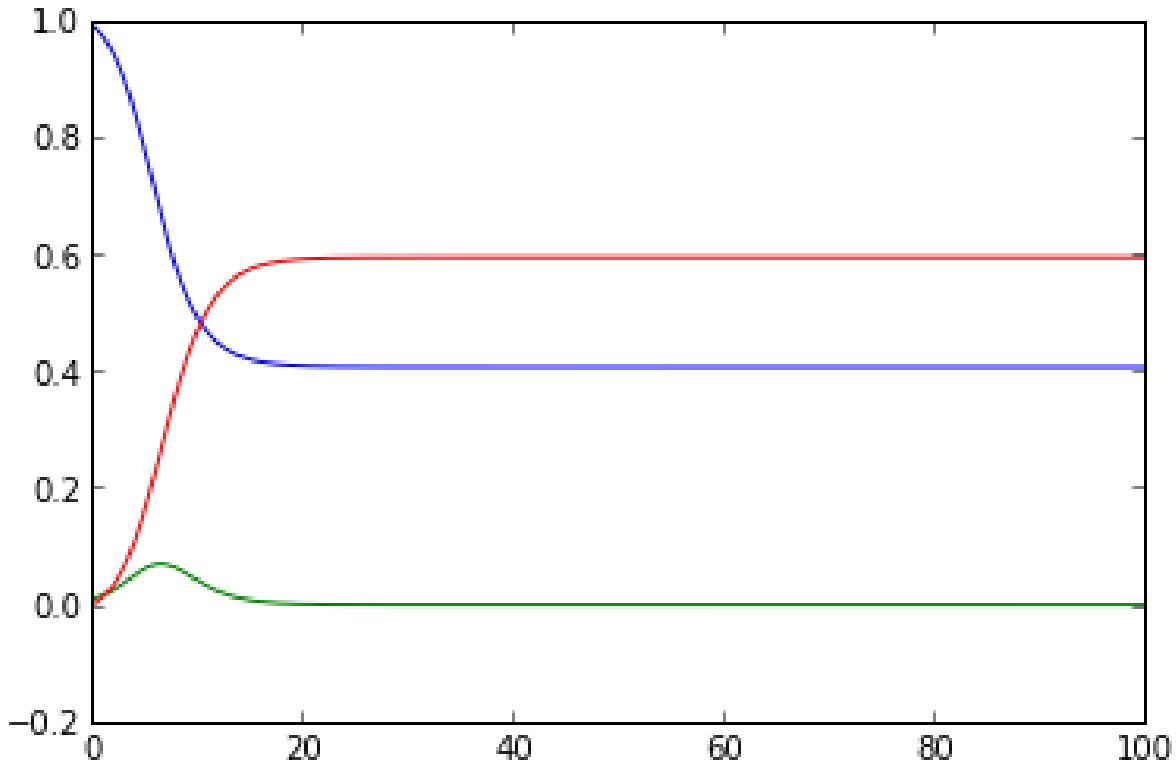
```
t = linspace(0,1000.0,1000)
plot(t,odeint(F(1),(0.99,0.01,0),t))
```

```
<matplotlib.lines.Line2D at 0x44d3510>,
<matplotlib.lines.Line2D at 0x44d3710>]
```



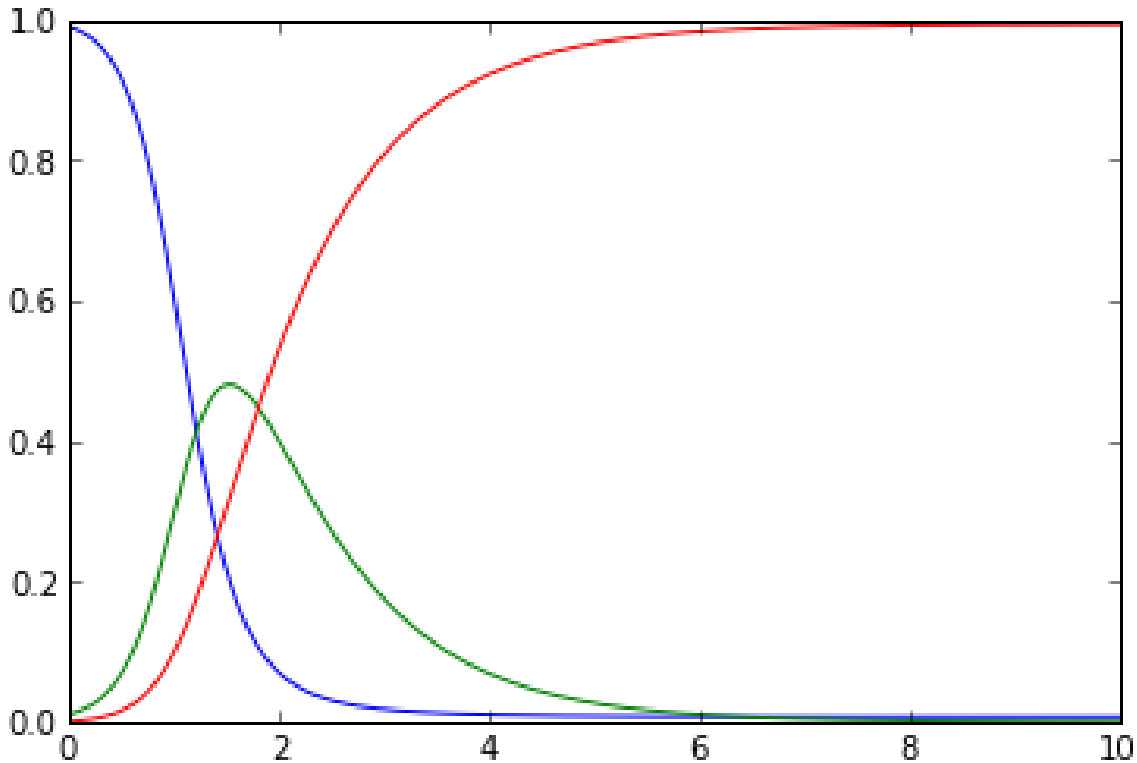
```
t = linspace(0,100.0,1000)
plot(t,odeint(F(1.5),(0.99,0.01,0),t))
```

```
<matplotlib.lines.Line2D at 0x45e9a50>,
<matplotlib.lines.Line2D at 0x45e9c50>]
```



```
1 t = linspace(0,10.0,1000)
2 plot(t,odeint(F(5.0),(0.99,0.01,0),t))
```

```
<matplotlib.lines.Line2D at 0x493fd50>,
<matplotlib.lines.Line2D at 0x493ff50>]
```



SIS Models

Some diseases have oscillations.

For that to occur, there has to be a constant supply of new susceptible individuals:

- disease don't confer immunity OR
- new births in the community

$$S \to I \to S$$

$$\dot{S} = -\beta SI + \gamma I$$
$$\dot{I} = \beta SI - \gamma I$$

Rescaling

$$\dot{X} = -\rho XY + Y$$

$$\dot{Y} = \rho XY - Y$$

Using $X + Y = 1$:

$= \mathrm{r}\ \mathrm{Y}\ (1\text{-}\mathrm{Y}/\mathrm{K})$

with $r = \rho - 1$ and $K = (\rho - 1)/\rho$

This is just the logistic regression, so it doesn't oscillate.

SIR Model with Birth

Constant birth/death rates

- μN is the birth rate
- $-\mu S$ etc. are the death rates
- they balance out
- note that R is not death
- equations constantly take individuals out of each population and put them back into S

$$\dot{S} = -\beta SI + \mu N - \mu S$$

$$\dot{I} = \beta SI - \gamma I - \mu I$$

$$\dot{R} = \gamma I - \mu R$$

Rescaling

$X = S/N$, $L = \beta I$, $S + I + R = N$

New equations (work this out yourself):

$$\dot{X} = \mu(1 - X) - LX$$

$$\dot{L} = (\gamma + \mu)L(\rho X - 1)$$

Jacobian

$$J(X,L) = \begin{pmatrix} \frac{\partial \dot{X}}{\partial X} & \frac{\partial \dot{X}}{\partial L} \\ \\ \frac{\partial \dot{L}}{\partial X} & \frac{\partial \dot{L}}{\partial L} \end{pmatrix} = \begin{pmatrix} -\mu - L & -X \\ \\ (\gamma + \mu)L\rho & (\gamma + \mu)(\rho X - 1) \end{pmatrix}$$

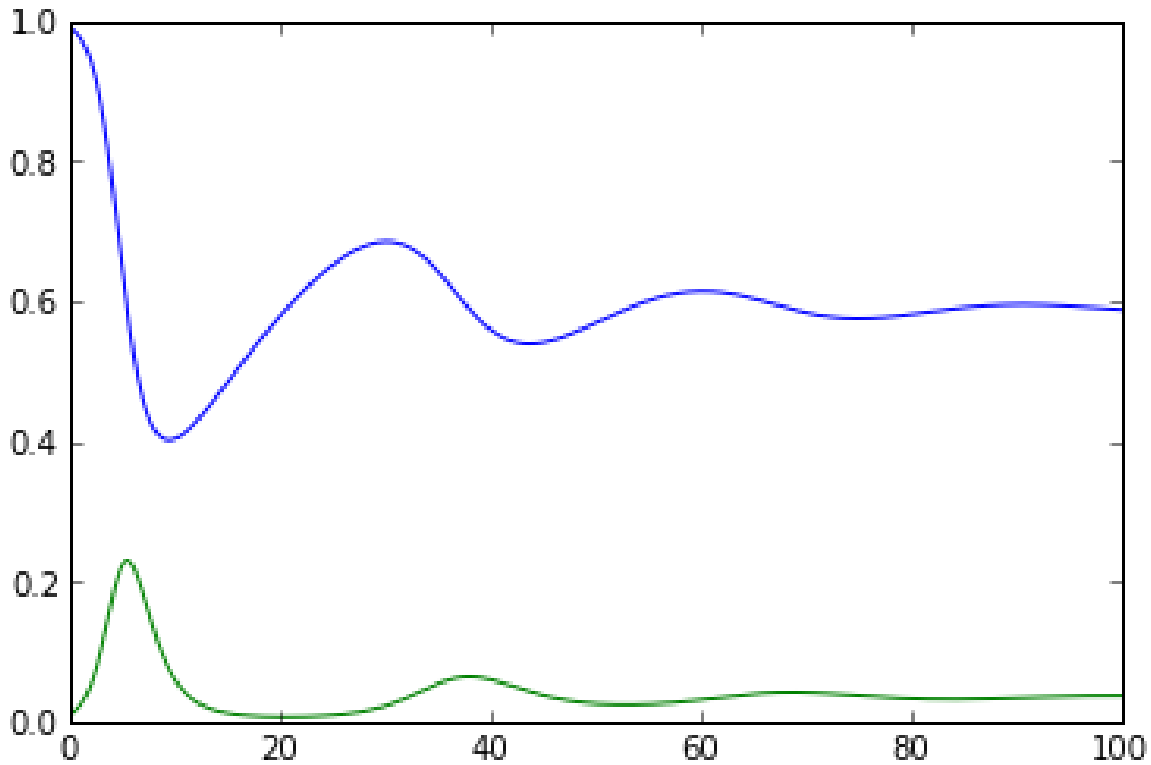
This gives us the linearized pattern of nullclines around some stable point.

- we can get stable steady states
- dampened oscillations towards a steady state
- continuous oscillations

```
1 def F(mu,rho,gamma):
2     def f(v,_): x,l = tuple(v); return(mu*(1-x)-l*x,(gamma+mu)*l*(
          rho*x-1))
3     return f
```

```
1 t = linspace(0,100.0,1000)
2 plot(t,odeint(F(0.05,1.7,1.2),(0.99,0.01),t))
```

```
<matplotlib.lines.Line2D at 0x4a5e850>]
```



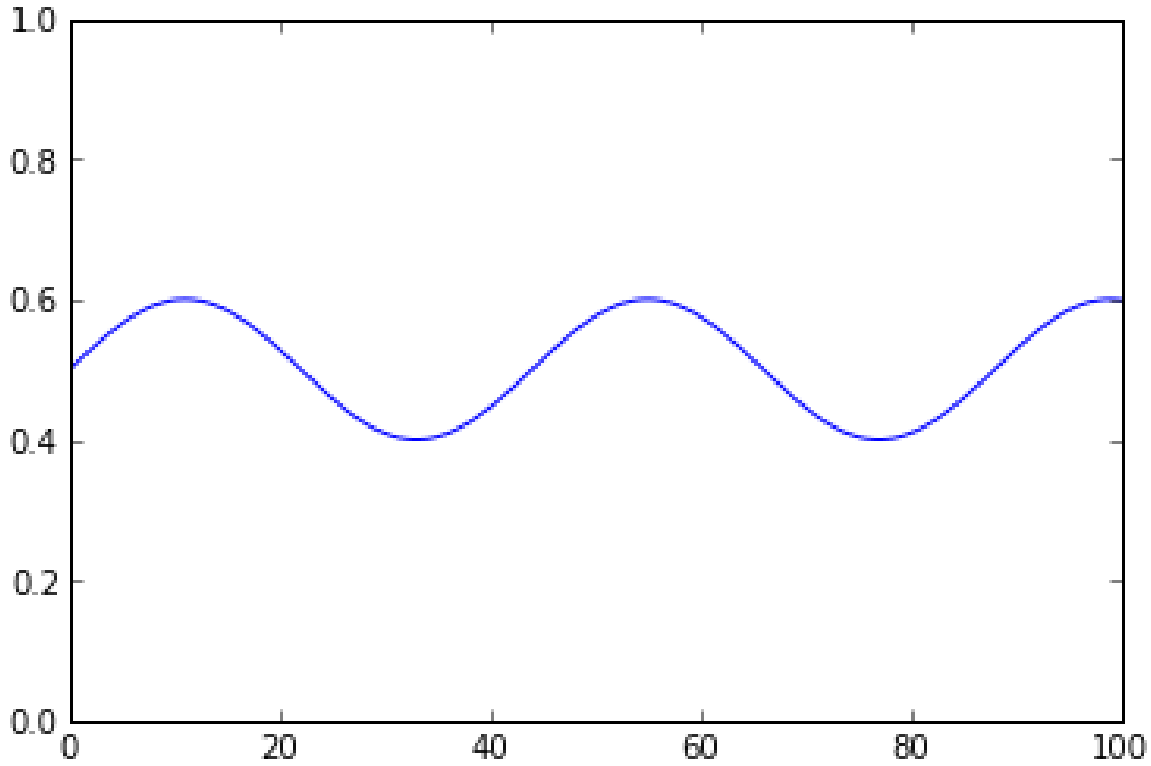
Parameter Fitting for Models

Next steps

- make reasonable guesses of ranges for μ , γ , β based on first principles
- try to fit parameters to match observed data

Let's just simulate some observations: a steady state of 50

```
1 ylim((0,1))
2 data = 0.5+0.1*sin(t/7.0)
3 plot(t,data)
```



We start parameter optimization by computing the error for a given set of parameters. Parameters are both initial conditions (x, l) and the actual parameters of the equation.

```
def error(params):
    mu,rho,gamma,x,l = tuple(params)
    pred = odeint(F(mu,rho,gamma),(x,l),t,rtol=1e-8,atol=1e-8)
    return sum((pred[:,0]-data)**2)
```

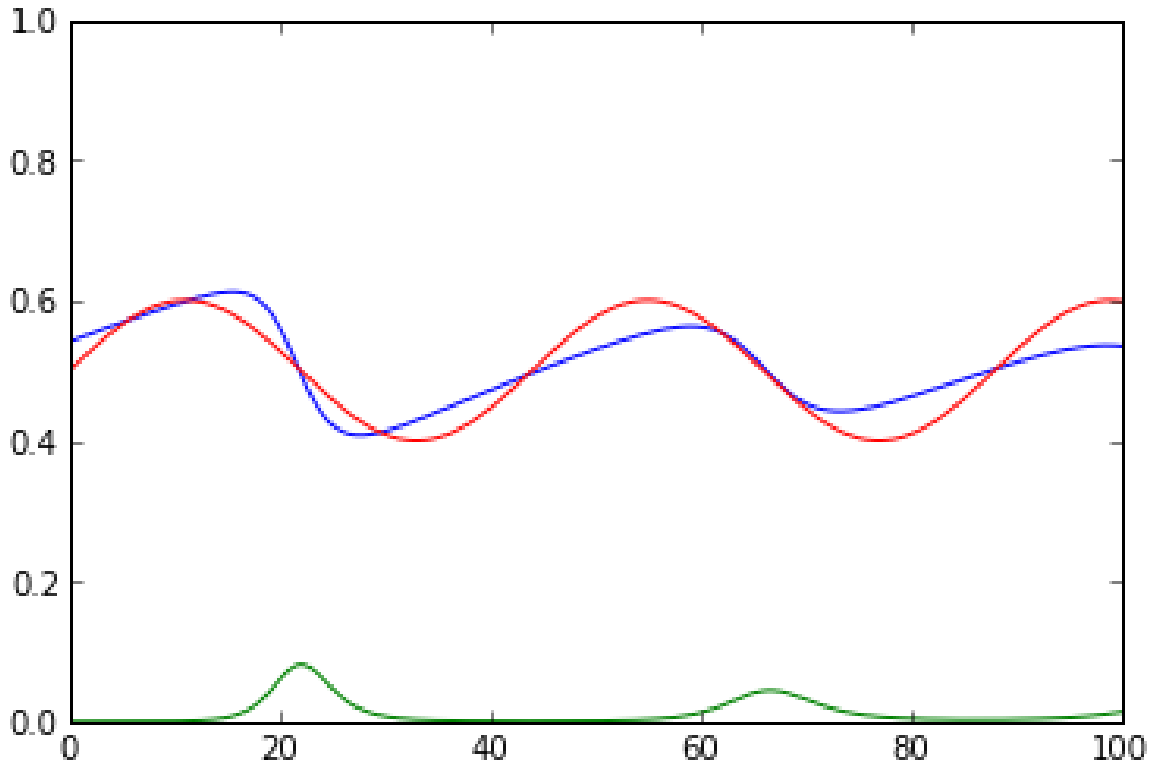
There are many different optimization algorithms possible. We prefer one that doesn't require computing derivatives. Nelder-Mead is a good and simple first choice.

```
1 from scipy.optimize import minimize
2 initial = (0.05,1.7,1.2,0.9,0.1)
3 result = minimize(error,initial,method='nelder-mead',options=dict(
      xtol=1e-4,disp=True,maxiter=10000))
4 print result
```

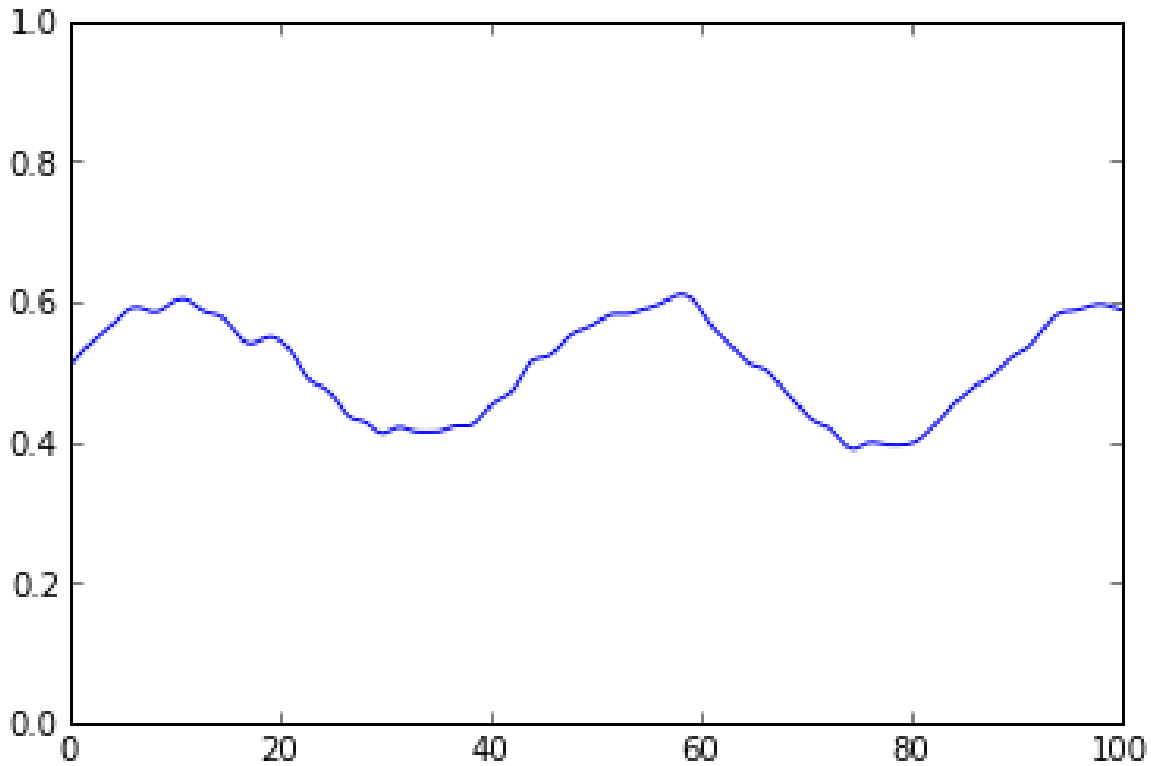
```
Optimization terminated successfully.
         Current function value: 1.046567
         Iterations: 599
         Function evaluations: 970
  status: 0
...
       x: array([  1.23405477e-02,   2.02340805e+00,   2.17068265e+00,
         5.41178930e-01,   2.00555648e-05])
 message: 'Optimization terminated successfully.'
     nit: 599
```

Optimization with Nelder-Mead gives us a reasonably good match to the data.

```
mu,rho,gamma,x,l = result.x
ylim((0,1))
plot(t,odeint(F(mu,rho,gamma),(x,l),t))
plot(t,data)
```



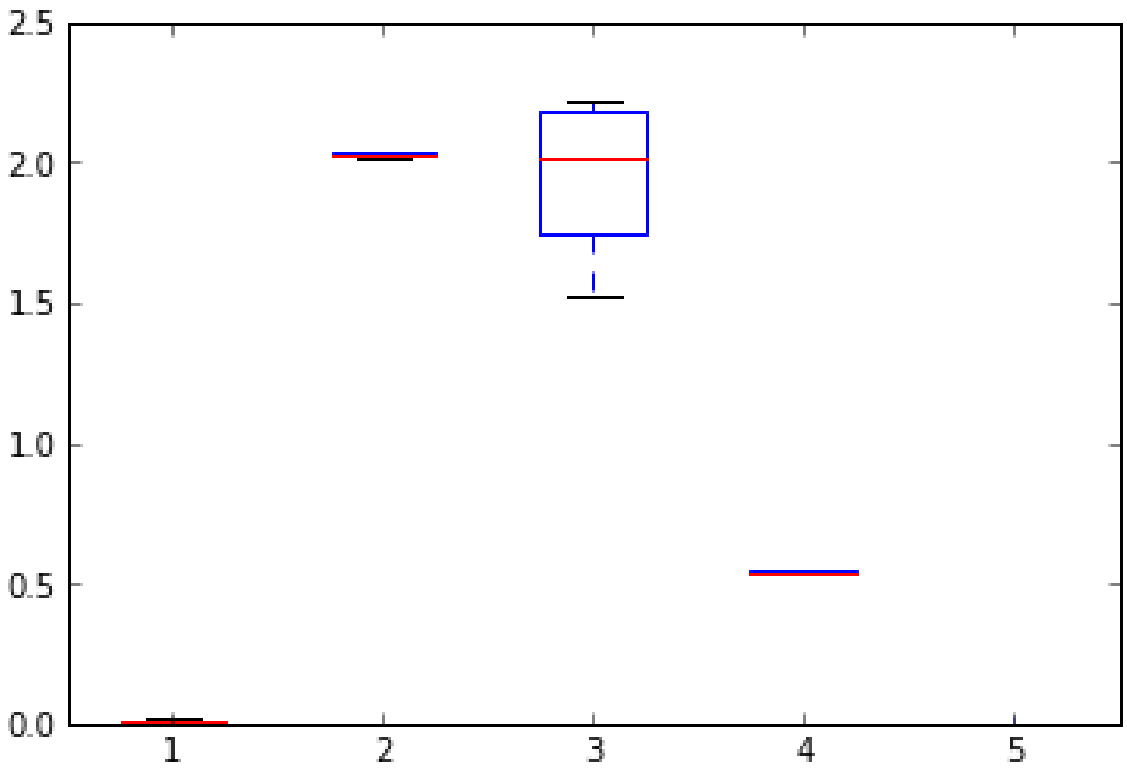
```
1 from scipy.ndimage import gaussian_filter as G
2 data0 = 0.5+0.1*sin(t/7.0)
3 ylim((0,1))
4 data = data0 + 0.05*G(randn(len(data0)),10.0)
5 plot(t,data)
```



```
results = []
for i in range(10):
    data = data0 + 0.05*randn(len(data0))
    initial = (0.05,1.7,1.2,0.9,0.1)
    result = minimize(error,initial,method='nelder-mead',options=
        dict(xtol=1e-6,disp=False,maxiter=1000))
    print i,result.x
    results.append(result)
```

```
Excess work done on this call (perhaps wrong Dfun type).
Run with full_output = 1 to get quantitative information.
Excess work done on this call (perhaps wrong Dfun type).
Run with full_output = 1 to get quantitative information.
0 [  1.47686975e-02   2.01836690e+00   1.60763901e+00   5.46792955e-01
...
8 [  1.21041822e-02   2.02991465e+00   2.21981275e+00   5.39325589e-01
   1.81061522e-05]
9 [  1.20469514e-02   2.02787696e+00   2.13171763e+00   5.44498268e-01
   2.66320564e-05]
```

```
1 xs = array([r.x for r in results])
2 _=boxplot(xs)
```



Disease? Computer Science?

Facebook vs Google

User growth

- If you know nothing else, what kind of curve would you predict for introducing a new service like Google+?
- How does growth actually work? How does it relate to the actual social network?
- (We'll return to the dimensionality of social graphs later.)

Facebook vs. Google Plus

$$\dot{I} = -\phi F - \gamma G$$

$$\dot{F} = \phi I - \epsilon F + \delta G$$

$$\dot{G} = \gamma G + \epsilon F - \delta G$$

Questions

- Assuming $G = 0.1F$ and some other parameters, can Google catch up at all?
- Assuming ϕ and γ are about the same, how much bigger does ϵ have to be than δ ?
- What are reasonable parameter values?
- Are oscillations possible?
- If users return to an inactive state at some rate λ , how does that change things?

Network Users

Network Users

- inactive users I become active at some rate
- active users A become inactive at some base rate
- network slowdown is proportional to $s = 1 - (A/M)^2$, where M is the maximum user population
- active users become inactive due to frustration at some rate $\phi(s)$, where $\phi(0) = 0$ and $\phi(1) = 1$

Questions

- Is this a reasonable model?
- Are oscillations possible?

Chemical Reactions and Hypercycle

Basic question of evolution

- Biological organisms are self-replicating networks of chemicals?
- How did complex networks evolve from simple networks?
- Similar networks in ecology and sociology.

$$A \to A$$

$$A \to B \to A$$

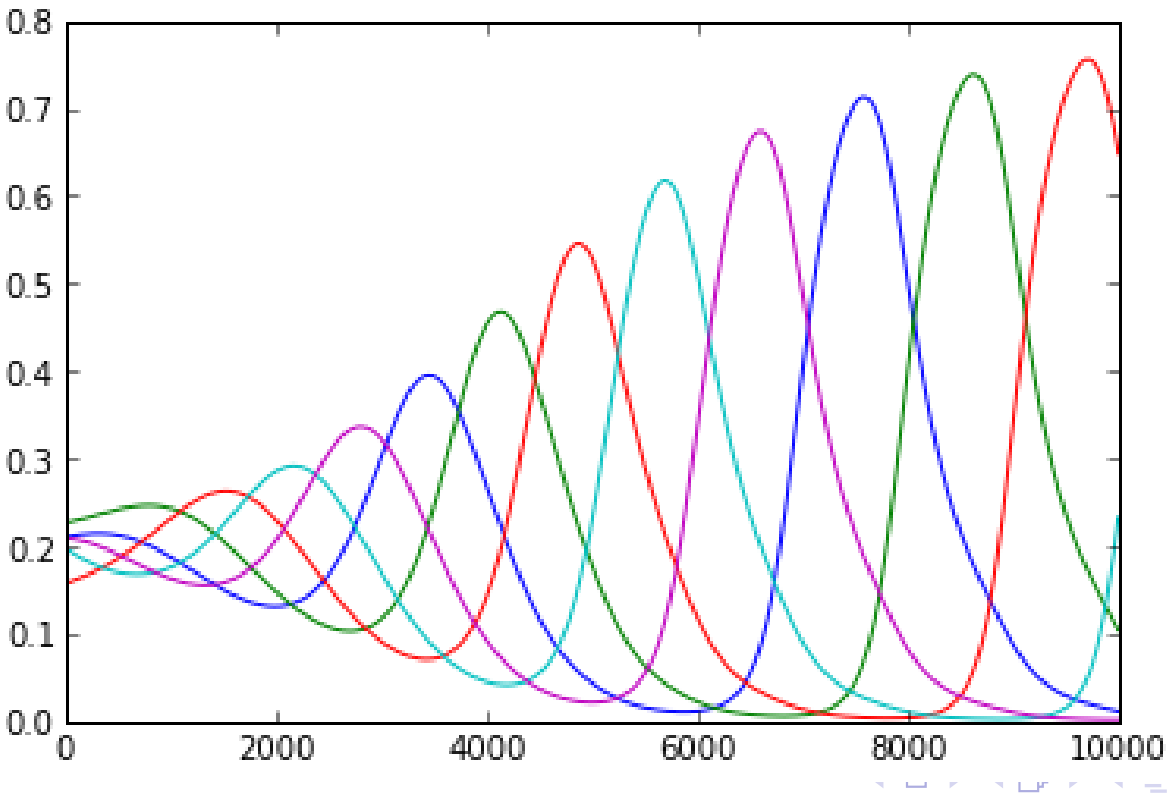
$$A \to B \to C \to A$$

Replication

$$A \to B \to C \to A$$

- B is replicated by A
- A is the *replicase*, B is the *template* (cf DNA)
- rate of replication is proportional to the concentration of A and the concentration of B

```
vs = []
v = 0.1*rand(5); v /= sum(v)
r = 0.01 * ones(5)
for i in range(10000):
    v += r*v*roll(v,1)
    v = v/sum(v)
    vs.append(v)
_=plot(vs)
```



Hypercycle results

Dynamics:

- such cyclic dependencies results in large oscillations over time

What happens when mutations arise?

- We would like it when better hypercycles outcompete worse hypercycles.
- Replicases that replicate other species better act as *altruists*, this is not rewarded.
- Chemical species that are better targets of replicases are rewarded by an increase in numbers.
- As is, better cycles do not outcompete worse cycles, evolution of complexity does not happen.

Solution:

- The unit of selection/competition needs to be the entire hypercycle.
- This is accomplished via compartmentalization or spatial localization.

Delay Differential Equations

Another form of differential equation that can be important in modeling real systems is delay differential equations.

Here is a simple example of a delay differential equation:

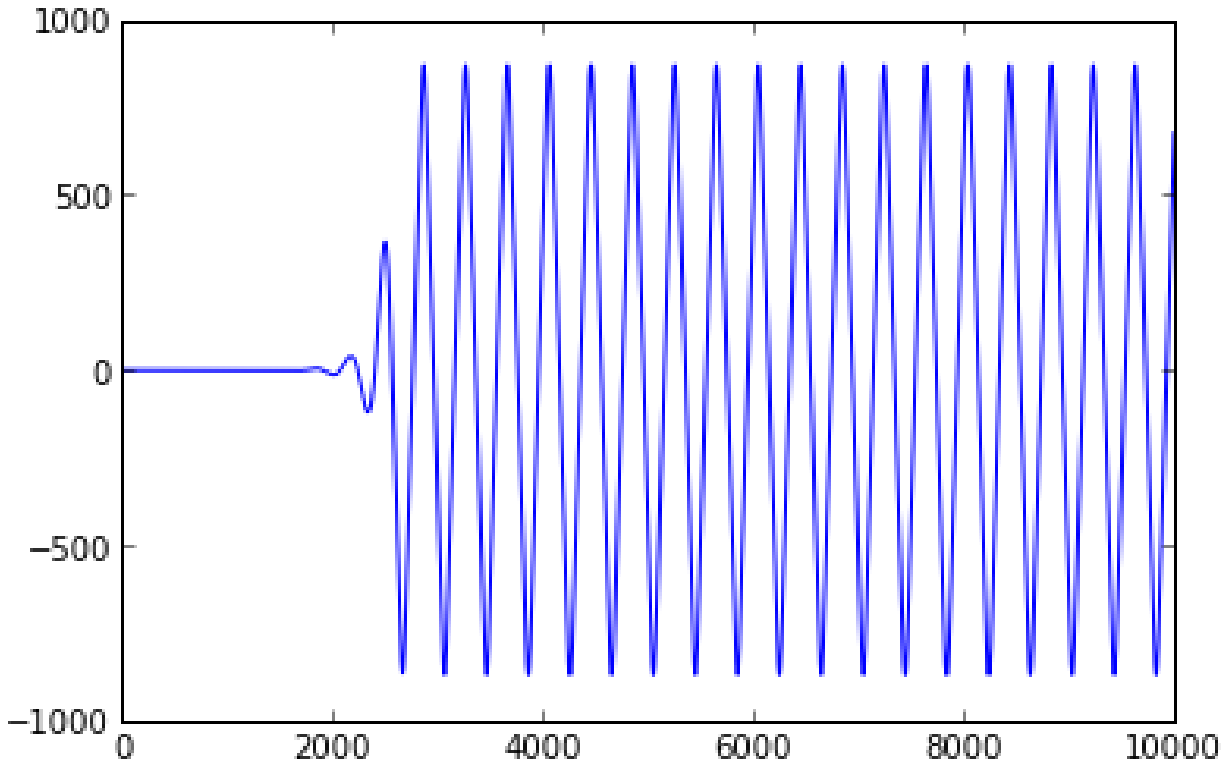
$$\dot{x}(t) = \alpha x(t - \tau)$$

Let's simulate this equation with a very simple difference equation.

```
alpha = -0.04
values = []
queue = 0.0001*randn(100); qi = 0
x = 0.0
for i in range(10000):
    values.append(x)
    x += clip(alpha*queue[qi],-10,10)
    queue[qi] = x; qi = (qi+1)%len(queue)
```

Note that delays in a differential equation tend to lead to oscillations even for very simple differential equations.

```
1 plot(values)
```



Very important example of delay differential equations: vehicle following, autonomous driving.

velocity $v_i(t)$, headway (distance to next vehicle) $h_i(t)$, reaction time τ

$$\dot{v}_i(t) = a_i(t)(V(h_i(t-\tau)) - v_i(t))$$

$$\dot{h}_i(t) = v_{i+1}(t) - v_i(t)\text{circular}$$

$$\sum h_i(t) = L$$

V is a control strategy.

- Control strategies using only $V(h_i(t-\tau))$ tend to be unstable
- Oscillations show up as traffic jams.
- Control becomes easier if you look one more car ahead $h_i(t)$, $\mathrm{h}_{i-1}(t)$