Random Covers

Problem:

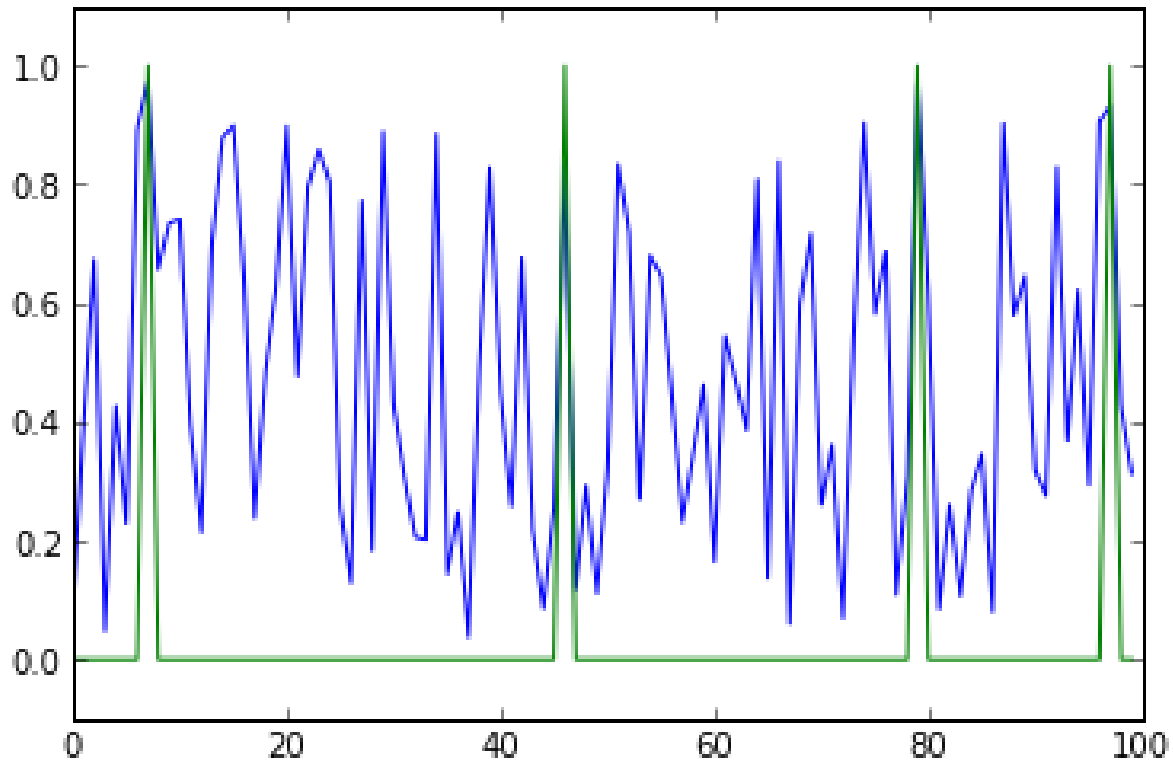
With only local information passing and processors at random locations, divide a 1D, 2D, or 3D region into subregions of approximately equal size.

Covering Protocol:

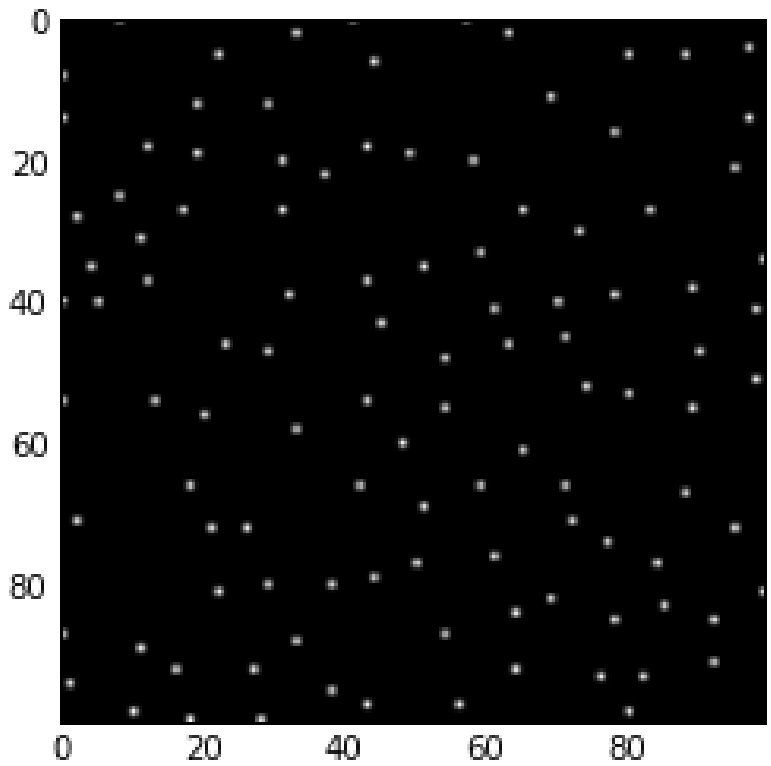
Goal: divide processors in a n -dimensional space into equal sized connected regions of size approximately d .

- every processor picks a random number
- processors broadcast their number to all neighbors within distance d
- processors become cell centers if they picked the maximum value among all their neighbors

```
a = rand(100)
b = filters.maximum_filter(a,20)
ylim((-.1,1.1)); plot(a); plot(a==b)
```



```
1 a = rand(100,100)
2 b = filters.maximum_filter(a,10)
3 imshow(a==b)
```



Problems:

- everything is noisy
- can't make perfect comparisons
- can't make a binary decision based on a threshold
- only nearest neighbor communications

Thresholding by Logistic Growth

Thresholding:

Idea: use logistic regression and growth rate r

$$\dot{u} = ru(1 - u)$$

If $r0$, this converges to 1 .

If $r0$, this converges to 0 .

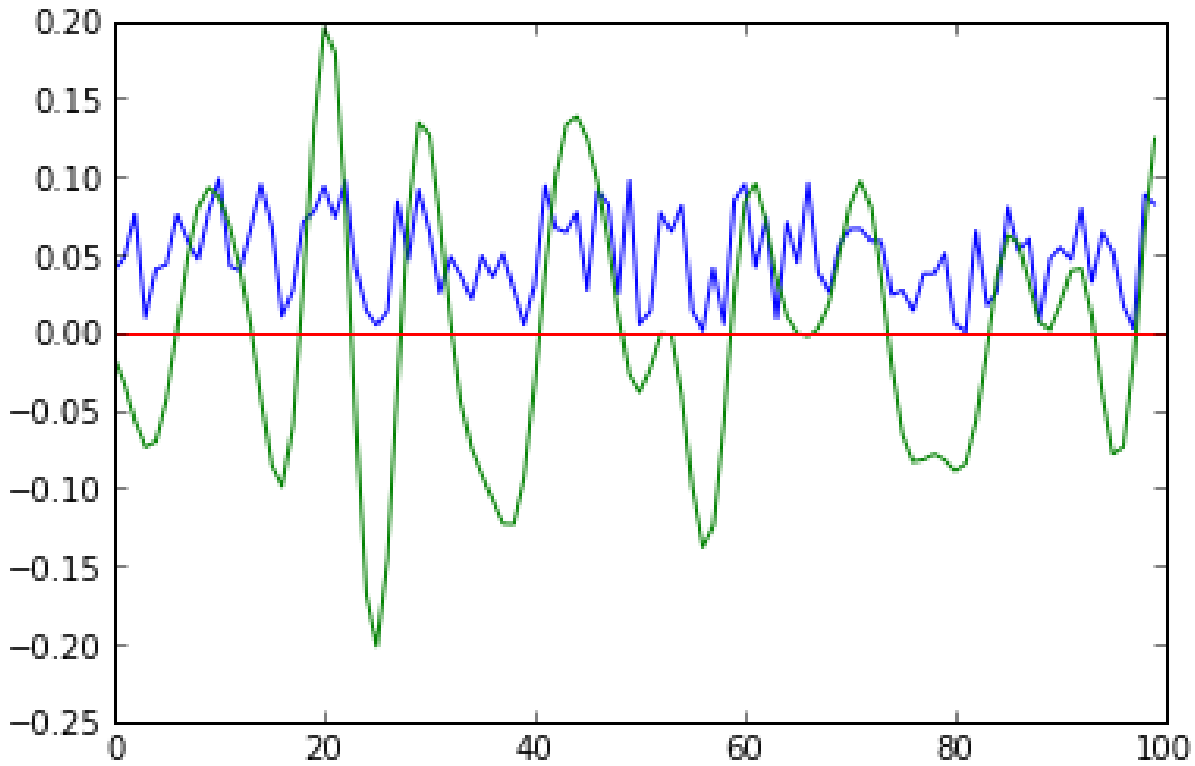
We can make r itself a variable.

Laplacian of Gaussian (or Difference of Gaussian):

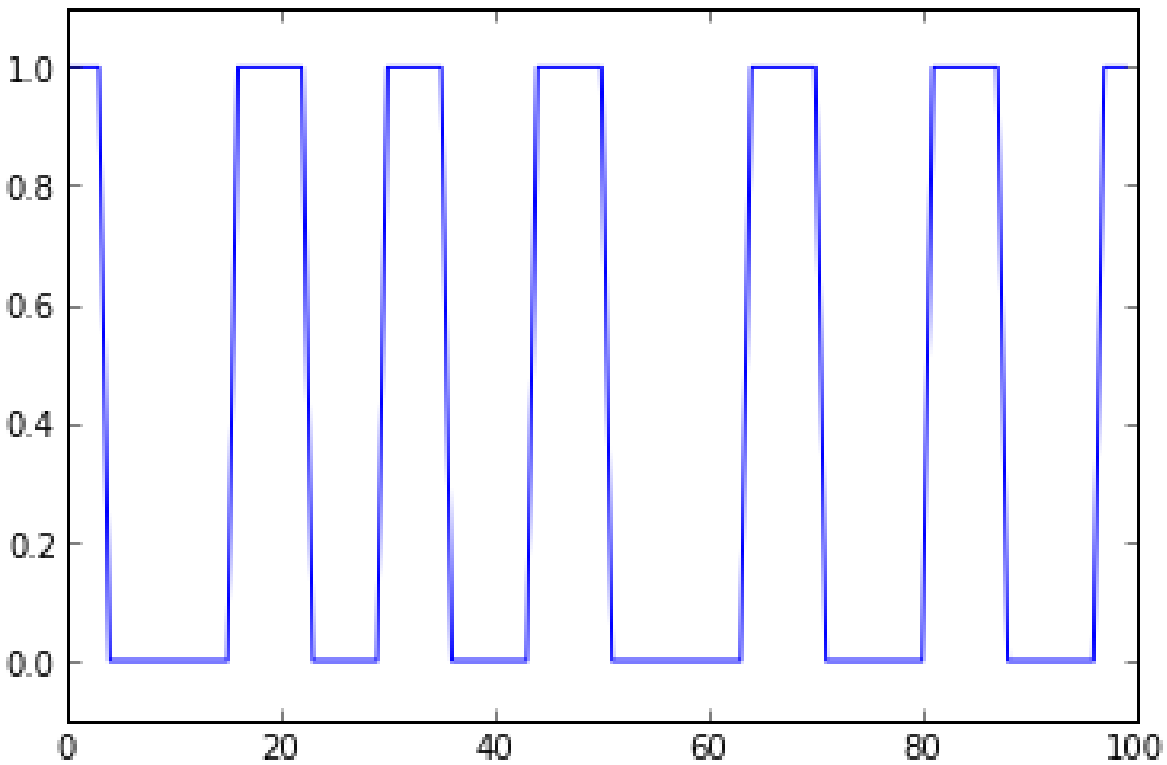
Use as the growth rate the Laplacian of Gaussian (LoG).

This will give rise to regions of positive and negative growth, spaced by about the size of the Gaussian.

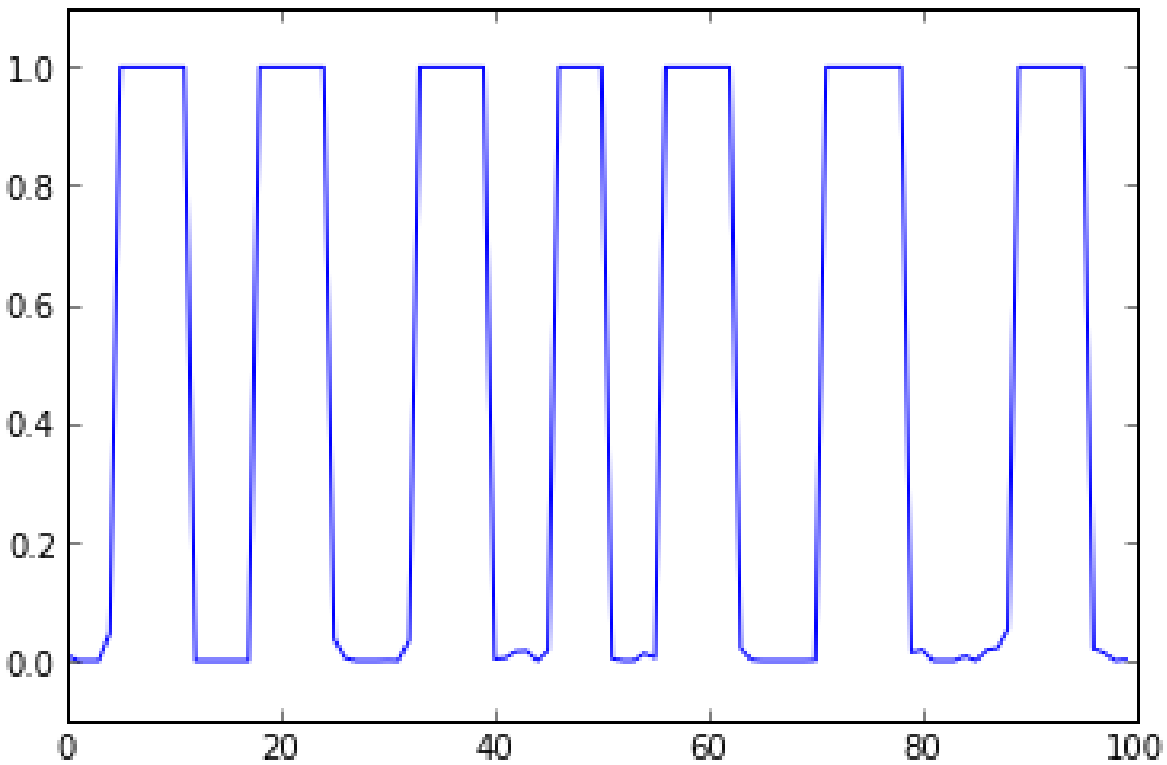
```
a = 0.1*rand(100)
r = -100*filters.gaussian_laplace(a,3.0)
plot(a); plot(r); plot([0,100],[0,0])
```



```
a = 0.1*rand(100)
for t in range(1001):
    r = -100*filters.gaussian_laplace(a,3.0)
    a = clip(a+r*a,0.001,0.999)
ylim((-.1,1.1)); plot(a)
```

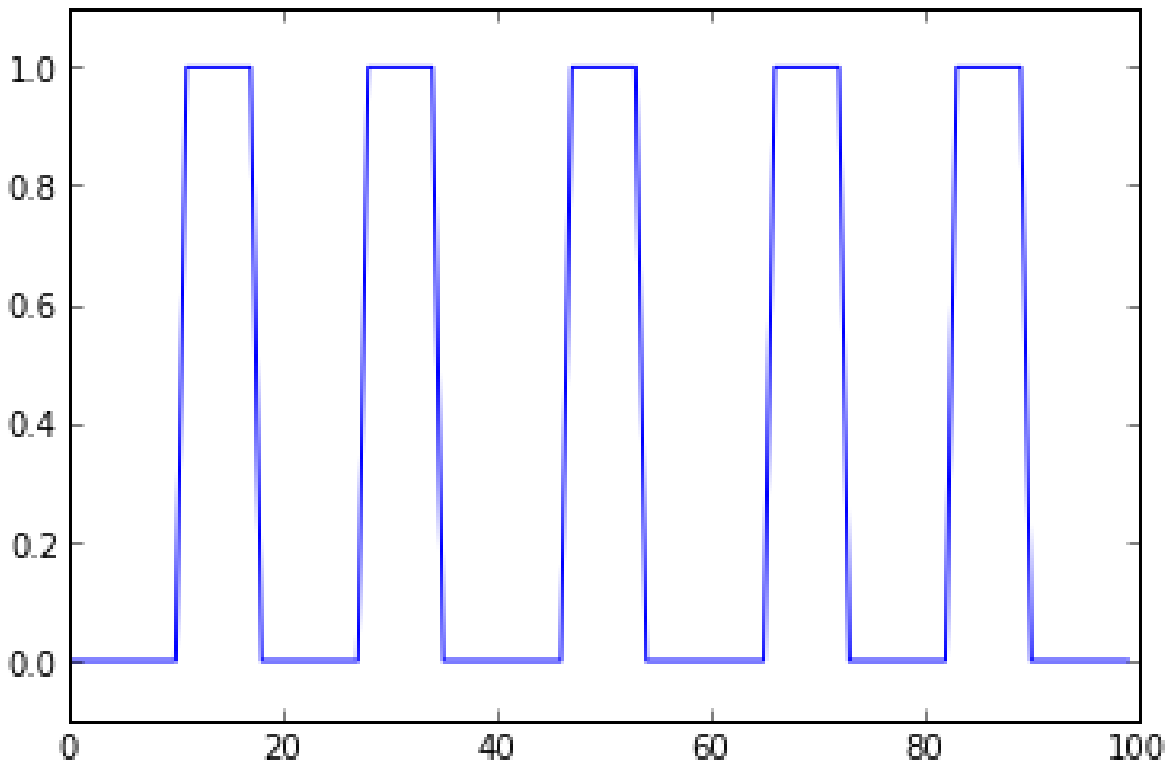


```
a = 0.1*rand(100)
for t in range(1001):
    r = -100*filters.gaussian_laplace(a,3.0)
    a = clip(a+r*a+0.03*rand(100),0.001,0.999)
ylim((-.1,1.1)); plot(a)
```

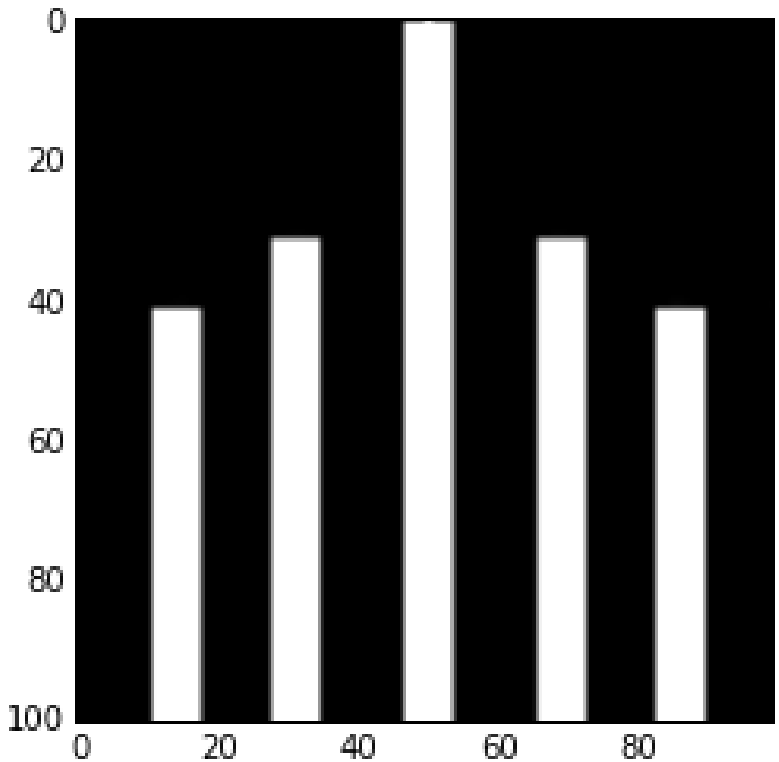


Wavefront Propagation

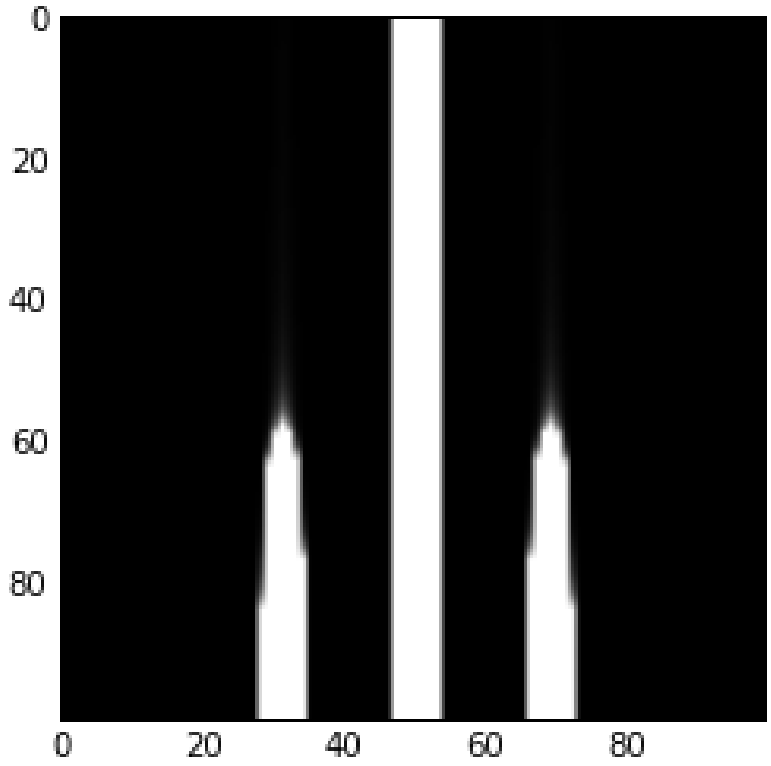
```
a = zeros(100)
a[len(a)//2] = 1
for t in range(10001):
    r = -50*filters.gaussian_laplace(a,3.0)
    a = clip(a+r*a,0.001,0.999)
ylim((-.1,1.1)); plot(a)
```



```
a = zeros(100)
a[len(a)//2] = 1
result = []
for t in range(10001):
    r = -50*filters.gaussian_laplace(a,3.0)
    a = clip(a+r*a,0.001,0.999)
    result.append(a)
result = array(result)
imshow(result[::100])
```



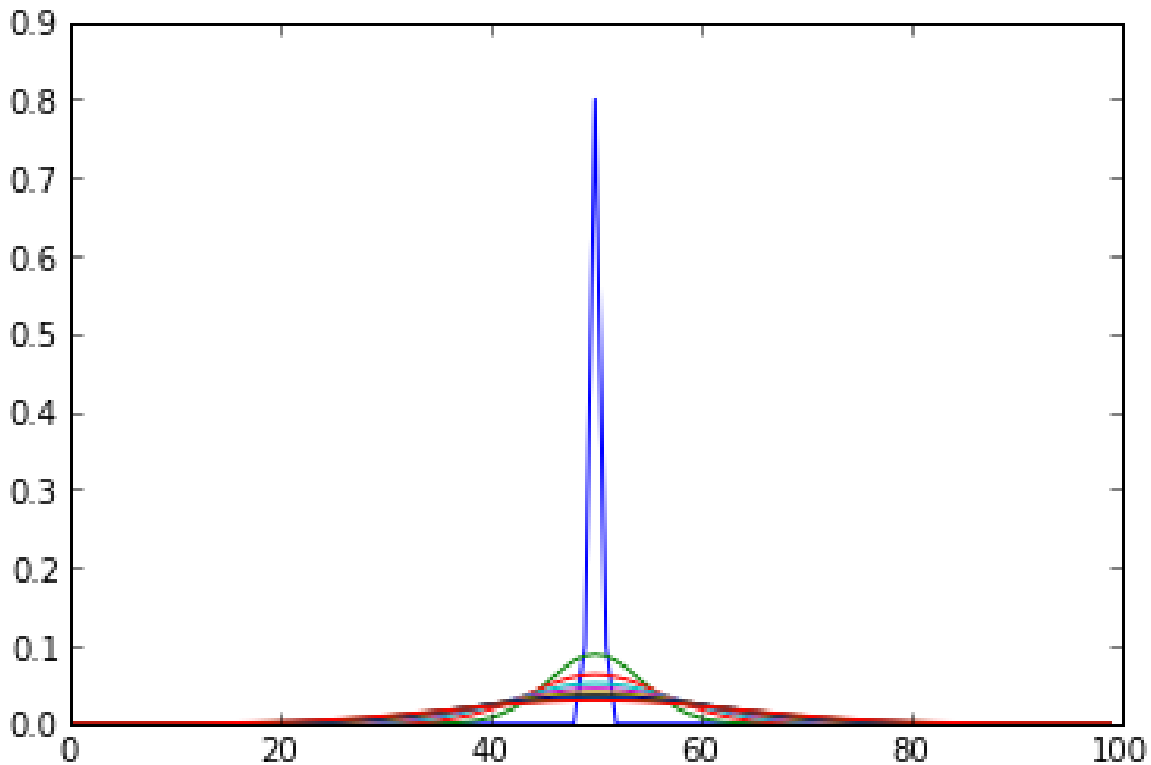
```
1 imshow(result[3100:3200,:])
```



Smoothing by Diffusion

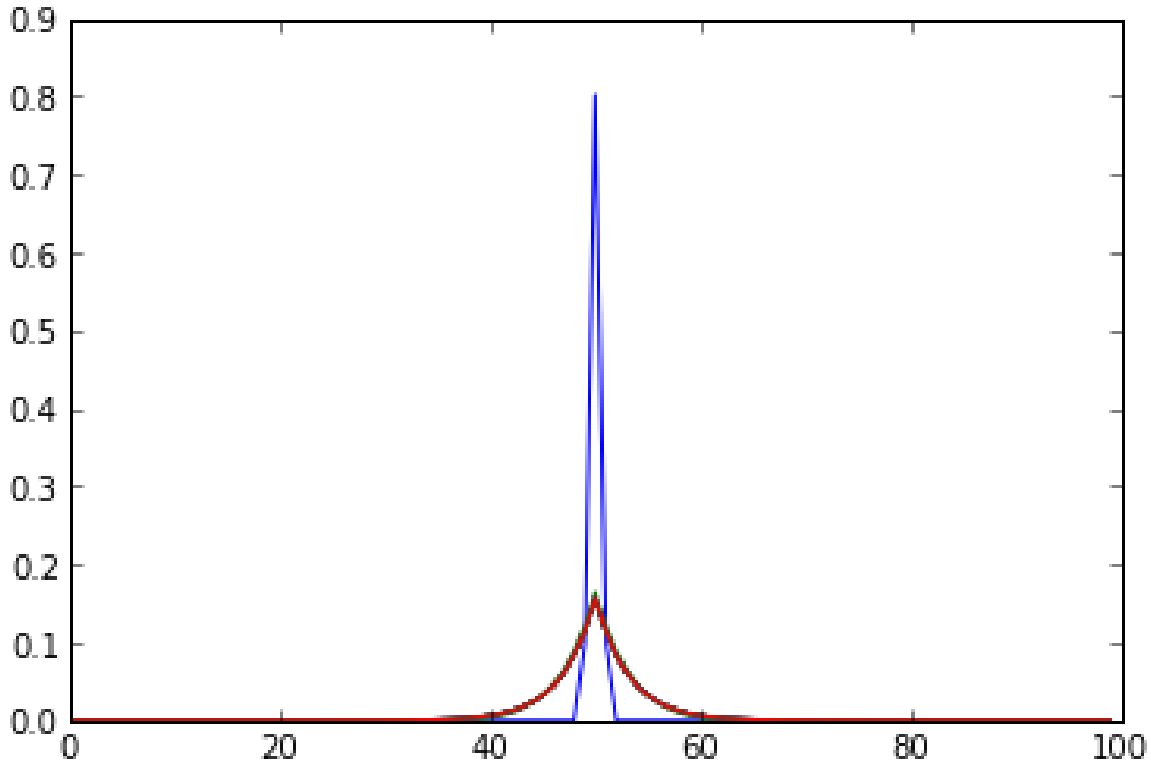
As we have seen, simple diffusion is like smoothing with larger and larger Gaussians over time.

```
a = zeros(100); a[len(a)//2] = 1
for t in range(1000):
    a += 0.1*laplacian(a)
    if t%100==0: plot(a)
```



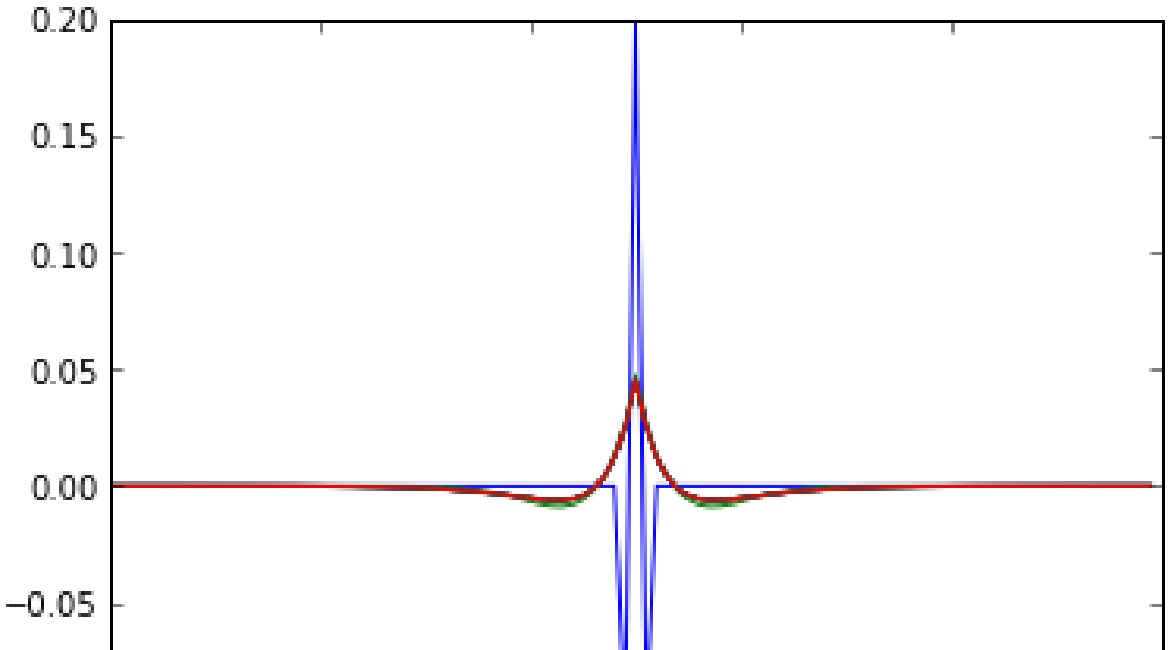
For the kind of center-surround inhibition we want for pattern formation, we need to construct a fixed degree of smoothing. We use the heat equation.

```
a = zeros(100); a[len(a)//2] = 1; b = a.copy()
for t in range(1000):
    b += 0.1*laplacian(b)
    b = 0.99*b+0.01*a
    if t%100==0: plot(b)
```



We can put together two of these now to obtain a center-surround filter.

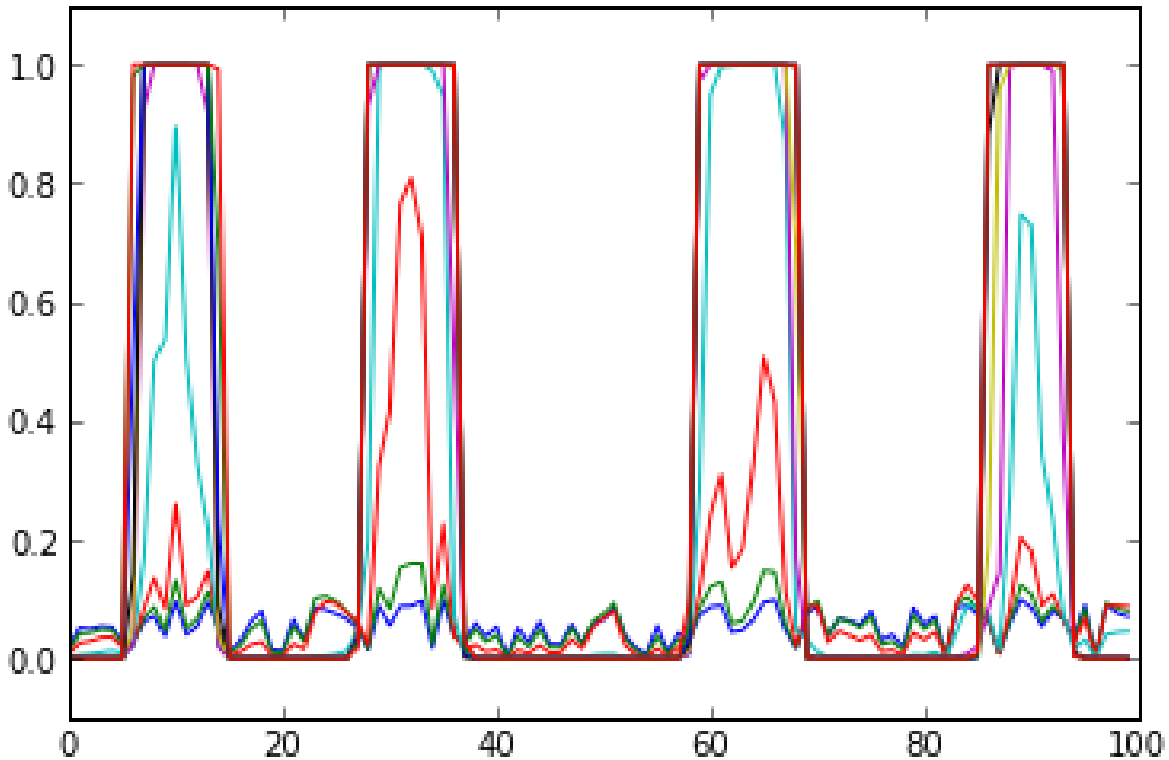
```
a = zeros(100)
a[len(a)//2] = 1
b = a.copy()
c = a.copy()
for t in range(1000):
    b += 0.1*laplacian(b)
    b = 0.99*b+0.01*a
    c += 0.2*laplacian(c)
    c = 0.99*c+0.01*a
    if t%100==0: plot(b-c)
```



Complete Reaction-Diffusion

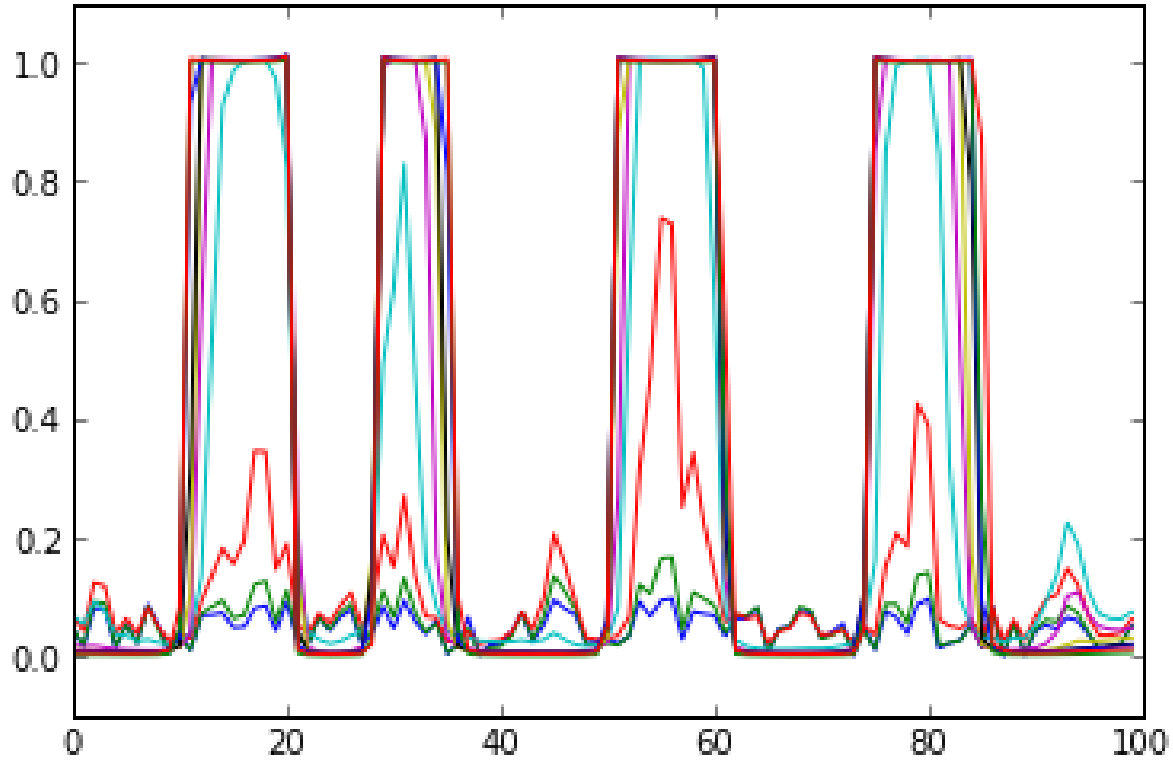
Now we use the two diffusions together with the logistic growth equation to arrive at a pattern formation equation.

```
a = 0.1*rand(100); b = a.copy(); c = a.copy()
for t in range(100000):
    b += 0.1*laplacian(b); b = 0.99*b+0.01*a
    c += 0.2*laplacian(c); c = 0.99*c+0.01*a
    a += 0.01*(b-c)*a*(1-a)
    if t%10000==0: ylim((-.1,1.1)); plot(a)
```



As before, we can add a bit of random growth to cause the stripes to shift closer together over time.

```
a = 0.1*rand(100); b = a.copy(); c = a.copy()
for t in range(100000):
    b += 0.1*laplacian(b); b = 0.99*b+0.01*a
    c += 0.2*laplacian(c); c = 0.99*c+0.01*a
    da = 0.01*(b-c)*a*(1-a); a += da + rand(len(a))*amax(da)*0.1
    if t%10000==0: ylim((-.1,1.1)); plot(a)
```



Mathematically, our reaction-diffusion equation now looks as follows.

$$\dot{b} = r_b \nabla^2 b + \lambda_b a - \gamma_b b$$

$$\dot{c} = r_c \nabla^2 c + \lambda_c a - \gamma_c c$$

$$\dot{a} = \gamma_a (b - c) a (1 - a) + \nu_a \epsilon$$

Here, ν_a and ϵ is an additional noise term (stochastic differential equation).

If the $\gamma_a \ll \gamma_b, \gamma_c$, then this behaves as originally designed:

- b and c are approximately smoothed versions of a
- the difference between b and c is a Laplacian-like center-surround operator
- each location tends towards 1 or 0 depending on whether it and its nearby growth rate is bigger or smaller than the intermediate average

Notes for the implementation:

- diffusion can be implemented with completely local communication
- diffusion can be implemented by randomly passing around a number of discrete tokens
- the exact from of the center-surround operator doesn't matter
- when γ_a is similar in magnitude to γ_b , we get complex dynamic behaviors (see in a minute)
- in most actual versions, variables b and c are combined

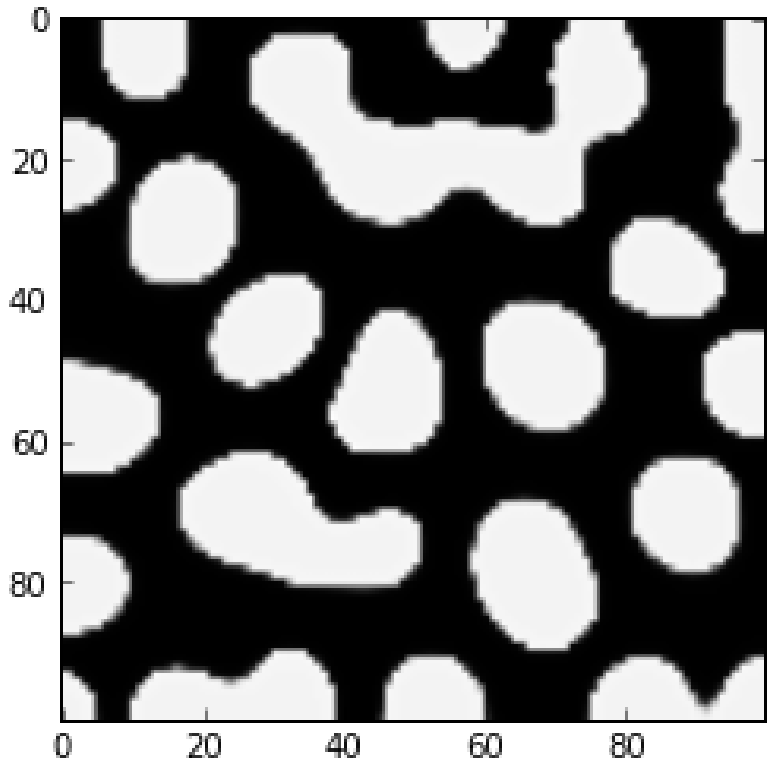
2D Pattern Formation

Everything works exactly the same way in 2D.

```
a = 0.1*rand(100,100); b = a.copy(); c = a.copy()
for t in range(100000):
    b += 0.1*laplacian(b); b = 0.99*b+0.01*a
    c += 0.2*laplacian(c); c = 0.99*c+0.01*a
    da = 0.01*(b-c)*a*(1-a); a += da + rand(len(a))*amax(da)*0.1
```

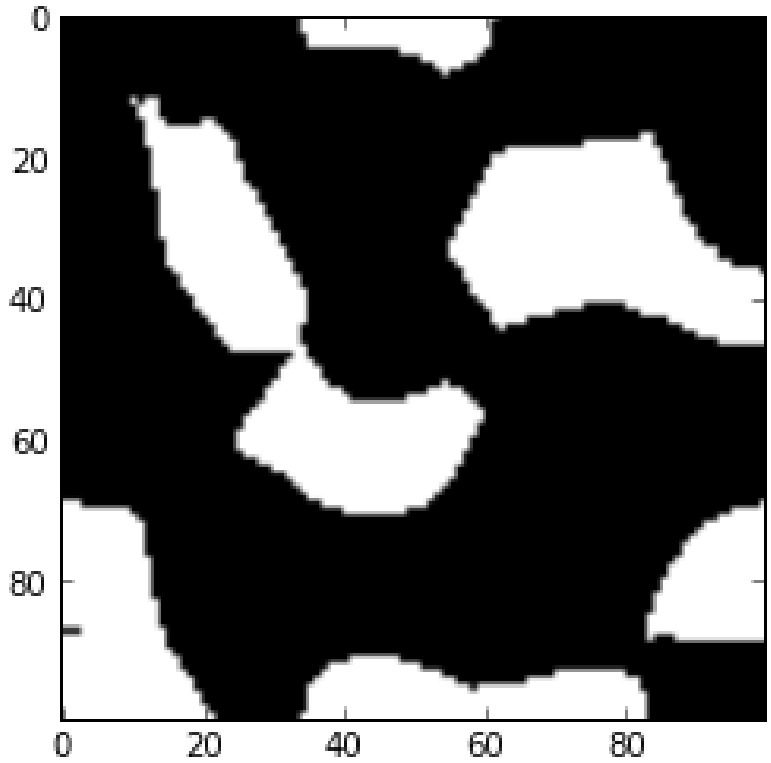
In 2D, we get spots (leopards) or stripes (zebras).

```
1 imshow(a)
```



Since we motivated pattern formation in reaction-dffusion systems as LoG smoothing followed by thresholding, what does that look like?

```
noise = randn(100,100)
gl = filters.gaussian_laplace(noise,12.0)
imshow(gl>scoreatpercentile(gl,70))
```



But, actually, we are effectively iterating the thresholding and smoothing multiple times, giving rise to this kind of iteration. This looks indistinguishable from reaction-diffusion patterns (for different parameters than above).

```
noise = randn(100,100)
gl = noise.copy()
for i in range(100):
    gl = filters.gaussian_laplace(gl,8.0,mode='wrap')
    gl = 1.0*(gl>scoreatpercentile(gl,50))
imshow(gl)
```

Summary

Reaction-Diffusion Systems:

Reaction-diffusion systems generate patterns by first smoothing noise, then amplifying and thresholding it.