Nonlinear Dynamics of Video Feedback

Timothy Burt
Department of Physics
Duke University
tburt@phy.duke.edu

Michail G. Lagoudakis
Department of Computer Science
Duke University
mgl@cs.duke.edu

1 Introduction

Feedback is one of the most direct routes to complicated nonlinear dynamics and chaos in practical applications. Sometimes the feedback process is beneficial, e.g. the mechanism by which the human retina adjusts to a range of illumination conditions. Other times of course the process is detrimental, for instance washing out some kind of signal we want to isolate or amplify. Using very simple equipment paired with computer modeling, we have specifically studied the dynamics of *video feedback*, which is important in contexts ranging from neural systems and image processing to graphics and art. Our observations are consistent with Crutchfield's findings from 1984 (see [1]), and we have furthermore demonstrated similar behavior using computational models.

2 Equipment and Procedure

In actuality, a very short list of basic items makes it possible to study a wide range of simple video feedback experiments: TV, video camera (or digital camera), computer with video capture ability (or interface to digital camera), and VCR. The first three of these components make up a closed loop: camera $\rightarrow$ computer $\rightarrow$ TV $\rightarrow$ camera. In our experiment, however, it proved impractical to attempt the computer interface: first, we could not locate a computer with video capture ability, and second we would not have been able to develop or obtain the software needed to modify and output the signals in real time. Therefore our feedback loop consists of only the two components, camera and TV, and in order to store the video information we simply connect a VCR.

We generate video feedback by aiming the camera directly at a portion of our TV's screen (with all background lights off or with moderate background illumination). We need only the most minute initial condition, such as a brief flash of light, to set the system in motion. This initial condition stimulates some region in the camera's receptive field, which causes a signal to be sent to the TV, which in turn alters the pattern displayed, and so on. The patterns respond to camera motion, including both translation and rotation. It is also possible to use the camera's focus and zoom features to change the dynamics in addition to controlling color, hue, brightness and contrast with the TV itself. Other means of perturbing the system include directly interfering with the receptive field of the camera (blocking some portion of the field with an object, or introducing an object in the field at a distance away from the camera). The final step is to connect our VCR to the TV output and record the feedback dynamics as it happens.

3 Observations

We have gleaned over six and a half minutes' footage of unique dynamical behavior from 1.5 hours of recorded video. In particular, we have found many examples of stable, persistent, pattern-forming behavior, including limit cycles, logarithmic spirals, and dislocations similar to those observed by

Crutchfield. Additionally, we discovered pulsating, organic shapes that refused to die away even under certain perturbations (including lateral camera translation, camera focus, and interference with the camera's receptive field).

We created the most interesting feedback patterns by rotating the camera about the axis passing through its lens: this proved to be one of the more difficult parts of the experiment, mostly because camera tripods do not typically permit this kind of rotation and because the feedback responds very rapidly to small changes in camera position. Despite the lack of precision inherent in manually adjusting the camera rotation angle, we reproduced strikingly stable limit cycles and rotating patterns, some of which slowed to a motionless or near motionless state as the camera angle changed (in most cases to $\pi/2$ radians).

4 Simulations

Without a computer in the loop, we could not reproduce initial conditions or apply well-controlled perturbations to the system, and our data consequently have not revealed many quantitative details. For this reason, we turned to computer simulation of the TV/camera system. We approached the problem in two ways, one method using a MATLAB calculation and the other running a similar calculation implemented in Perl/Tk. In the next sections, we will discuss the details of these simulations and the features they have in common with the actual video feedback observed.

4.1 Model

A simple computational model of the experimental setup has been derived. Both the camera and the TV screen are modeled as square matrices of the same dimension[1]. Let $Camera$ be the camera matrix and $Screen$ be the TV screen matrix. Both have dimension $2n+1$ where n is an integer number. Elements in both matrices are indexed from the middle element which corresponds to the origin $(0,0)$.

In order to simulate the dynamics of the actual system, we assume that the camera and the TV screen matrices are not necessarily aligned, but there is displacement of their centers and an angle between their axes. Let (d_x, d_y) be the displacement amd θ the angle. Obviously, by changing these parameters different mappings are achieved. For fixed values, a mapping M from camera elements to screen elements (and vice-versa) is well-defined. Notice that due to the finite size of the matrices some camera elements are probably mapped outside of the screen. The mapping M is defined formally below. Notice the use of the round operator to obtain integer values.

$$M : (i,j) \to (i',j'), \quad \begin{pmatrix} i' \\ j' \end{pmatrix} = \text{round}\left(\begin{pmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{pmatrix} \times \begin{pmatrix} i \\ j \end{pmatrix} - \begin{pmatrix} d_x \\ d_y \end{pmatrix} \right)$$

Light intensity at each camera or screen cell is represented by a real number between 0 and 1, 0 meaning absolute darkness and 1 maximum brightness. It is assumed that the background surrounding the TV screen is dark.

We further assume that the light intensity at each camera photoreceptor, that is, at each element $Camera(i,j)$ in the camera matrix, is a (nonlinear) function F of the light intensity of the screen elements within a circular area $RCF(i,j)$, called the receptive field. This circular area is centered at a reference point $Screen(i',j')$ on the screen matrix, whose position is uniquely determined by the mapping $M : (i,j) \to (i',j')$ described above. Let r be the radius of the receptive field. Then the camera elements are updated as follows:

$$Camera(i,j) = F(RCF(i,j))$$

$$RCF(i,j) = \left\{ Screen(i'',j'') : \quad \left\| \begin{pmatrix} i'' \\ j'' \end{pmatrix} - \begin{pmatrix} i' \\ j' \end{pmatrix} \right\|_2 \leq r, \quad \begin{pmatrix} i' \\ j' \end{pmatrix} = M\left(\begin{pmatrix} i \\ j \end{pmatrix} \right) \right\}$$

[1] This is just for simplicity. In the opposite case, appropriate scaling would be necessary.

Finally, once all camera elements are updated, they are copied on the screen.

$$Screen(i,j) = Camera(i,j)$$

Notice that this implies parallel update of the screen cells. Although sequential update would be more plausible, we preferred the simplicity of the parallel one that allows for observations of dynamic behavior uniformly over the whole screen.

The nonlinear function F can take several forms. We have implemented and tried the following:

Average This is simply the average intensity of the elements within the receptive field. Although this is not a nonlinear function, it served as our starting point. In fact, for the nonlinear mappings described below, the average is calculated first (it falls in the range $[0,1]$) and then is mapped nonlinearly in the interval $[0,1]$. Using just the average for F results in a smoothing of light intensity over the screen at the medium level, as expected.

Sigmoid This is a sigmoid function shifted by 0.5 to the right because of our [0,1] values.

$$g(x) = \frac{1}{1 + e^{-2*\beta*(x-0.5)}}$$

The constant β determines the slope of the sigmoid at $x = 0.5$. With this choice for F and a radius $r \geq 1$ our simulation in effect implements a simple Hopfield neural network model with lateral excitatory connections. The effect of the sigmoid function is that high light intensity is strengthened, whereas low intensity is suppressed even more.

ON-Center Cells This dynamics was inspired by the ON-Center ganglia cells found in the human retina. These neurons have a circular receptive field sensitive to light, where both excitation and inhibition are present. The inner-most (central) photopigments excite the neuron (and hence, ON-Center), whereas the outer-most ones inhibit the neuron. The average light intensity of the cell within $1 < \rho < 1.5$ (inhibition) is subtracted by the average intensity of the cells within $\rho \leq 1$ (excitation). The result passes through a regular sigmoid function ($g(x) = 1/(1+e^{-2*\beta*x})$) before updating the cell, to guarantee that values remain within the $[0,1]$ range.

Logistic Map ($f^1(x)$) This is the well-known logistic map

$$x_{n+1} = f^1(x_n) = \rho x_n(1 - x_n)$$

where $0 \leq \rho \leq 4$.

Double Logistic Map ($f^2(x)$) This is the logistic map applied twice

$$x_{n+1} = f^2(x_n) = f^1(f^1(x_n)) = \rho^2 x_n(1 - x_n)(1 - \rho x_n(1 - x_n))$$

4.2 Implementation

The model was implemented using the MATLAB mathematical software environment. The actual code is given in the appendix. On one hand MATLAB provides a lot of functionality and a set of implemented functions that allow for rapid prototyping. However, this comes at the cost of speed. MATLAB code is generally slow in execution. Although speed is an important factor in our case if we want to achieve video feedback simulation, it was more important for us to assess the validity of the model and the visualization tools of MATLAB were the easiest and fastest way to do it. The Perl/Tk implementation was an attempt to speed up the animation however graphical manipulation was much more difficult.

The current implementation sets the initial state of the system using a uniform (pseudo)random number generator (corresponding to screen noise). Other initializations have not been explored here.

4.3 Experimental Runs

The first run shown in Figure 1 made use of the sigmoid function without any displacement or angle. The matrices here and in all subsequent examples are of size 101×101. The constant β of the sigmoid was taken equal to 3. The initial light distribution was taken to be random. Snapshots are shown at steps 0, 5, 10, 20. It seems that the system attempts to aggregate together blobs of high and blobs of low activation.

The second simulation shown in Figure 2 employs the exact same dynamics with the difference that there is a displacement of $(3, 5)$ cells and an angle of $\pi/6$. Notice that the same short of pattern emerges centered in the middle though due to the resulting rotation.

The third simulation shown in Figure 3 employs the logistic map. In our experiments we observed that for $\rho \leq 1$ light intensity dies out, whereas for $1 < \rho \leq 3$ it goes to certain level in the scale, as expected. The most interesting (chaotic) behavior is for $3 < \rho \leq 4$ where the light intensity cannot settle down to any steady state, not even oscillate between a finite number of levels. The whole pattern bounces back and forth between high and low intensities without continuity in time, but with some continuity in space (pattern). Figure 3 shows six snapshots of a run with $\rho = 3.7$ at steps 0, 5, 10, 18, 19, 20. Notice the difference between consecutive updates. There is no displacement or angle between the matrices.

The last example employs ON-Center cell dynamics. β was taken equal to 2 in this run. It is interesting how the string pattern emerges essentially from chaos.

4.4 Limitations

Our model is a very crude approximation of the real dynamics. Firstly, it does not scale easily to the regular TV screen resolution and operation in real-time is out of question. Another limitation is that at the moment it does not support magnification (zoom). This feature would add many more capabilities to the model. There are also other features that are not currently modeled (camera focus, screen brightness and contrast, etc.) that would enhance the model if added. Finally, initialization at any arbitrary state, instead of random, would be desirable. However, even with these limitations the model is powerful to produce complex dynamics as demonstrated above.

5 Discussion

The simple experimental setup we used implements an inexpensive, fast and massively parallel simulator of spatiotemporal dynamics (2D cellular automaton, 2D neural map). The spatial dimension appears in the 2D patterns formed, whereas the temporal dimension involves the evolution of those patterns in time. The numerous parameters of the system allows for different dynamics and demonstrated behavior. However, the time unit of the system is probably one feature that is pretty much fixed by the hardware of the system. It basically depends on the rate of scanning of the TV and the camera. Nevertheless the richness and the sensitivity of the resulting behavior provide ground for experimental creativity and imagination.

What we learned from this project is that the complexity and power of common devices should never be underestimated. Our hands-on (actually, eyes-on) experience was a time of creativity and excitation. Every new behavior that was discovered during the experiments, attracted our attention and will remain vivid for a long time.

6 References

1. Crutchfield, J. P. "Space-Time Dynamics in Video Feedback", *Physica D*, **10**, pp. 229–245 (1984).

2. Hausler G., Neumahr T., Schonfeld H., Spellenberg B. "Chaos, order, and associative memory in video-feedback", Proc. SPIE 2039, *Chaos in optics*, San Diego (1993).

Appendix A: MATLAB Code

```
% NonLinear Dynamics of Video Feedback
% By Michail G. Lagoudakis and Timothy C. Burt

% PARAMETERS
%
% "n" is integer 2n+1 will be the size of the matrices
% "max_steps" is the maximum number of iterations
% If "scale" is 'y' the colorbar will remain scaled to [0,1]
%    otherwise will scale according to the data
% If "par" is 'y' the screen is updated only after all camera cells
%    have been updated (parallel updating), otherwise each screen cell is
%    updated right away.
% "rho" determines the size of the receptive field
% "displacement" is a (2x1) array that defines the displacement of the matrices
% "angle" (in rads) defines the angle between the axes


function video(n, max_steps, scale, par, rho, displacement, angle)

%%%%%%% Default Values

if nargin<7, angle=0; end
if nargin<6, displacement=[0;0]; end
if nargin<5, rho=1; end
if nargin<4, par='y'; end
if nargin<3, scale='y'; end
if nargin<2, max_steps=10; end
if nargin<1, n=20; end


%%%%%%% Initialization

S = rand(2*n+1);
C = zeros(2*n+1);

if scale=='y', imagesc(S, [0 1]); else imagesc(S);  end
axis image; colorbar; drawnow;


%%%%%%% Main Loop

for step=1:1:max_steps

tic

for cameravx=-n:1:+n
for cameravy=-n:1:+n
```

```
   camerav = [cameravx; cameravy];

   rot = [ cos(angle) sin(angle) ; -sin(angle) cos(angle) ];
   screenv = round(-displacement + rot * camerav);

   sum = 0.0;
   count = 0.0;

   for pointx = screenv(1)-ceil(rho):1:screenv(1)+floor(rho)
      for pointy = screenv(2)-ceil(rho):1:screenv(2)+floor(rho)

          point = [pointx; pointy];

          if (-n<=point) & (point<=n) & (norm(point - screenv)<=rho)
             sum = sum + S(point(1)+n+1, point(2)+n+1);
             count = count + 1;
           end
      end
   end

   if count~=0, temp = (sum)/count;
   else temp=0; end


   %%%%%%%% Choose Dynamics

%    update = temp;                              % Average
    update = 1/(1+exp(-2*3*(temp-0.5))); % Sigmoid
%    update = f(temp); % f1(x)
%    update = f(f(temp)); % f2(x)
%    ON-Center in different file


   %%%%%%%% Parallel or Sequential Update

   if par=='y', C(camerav(1)+n+1, camerav(2)+n+1)=update;
   else  S(camerav(1)+n+1, camerav(2)+n+1) = update; end

end
end

if par=='y', S = C; end


%%%%%%%% Draw Screen

if scale=='y', imagesc(S, [0 1]); else imagesc(S);  end
axis image; colorbar; drawnow;

toc
```

```
end

return;


%%%%%% Logistic Map

function res=f(x)

r=3.7;
res = r*x*(1-x);
```

Appendix B: Figures

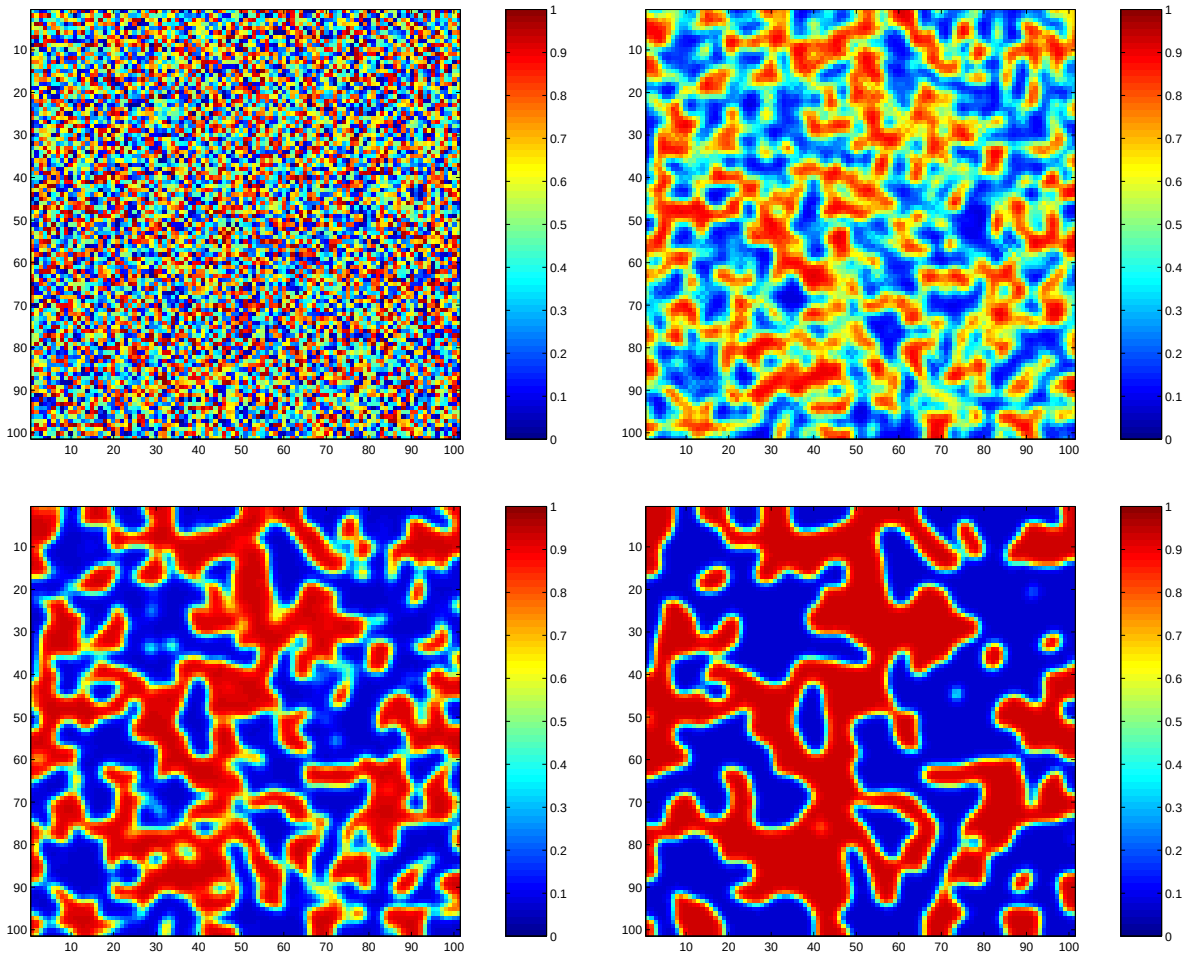

Figure 1: Sigmoid dynamics shown at steps 0, 5, 10, 20.

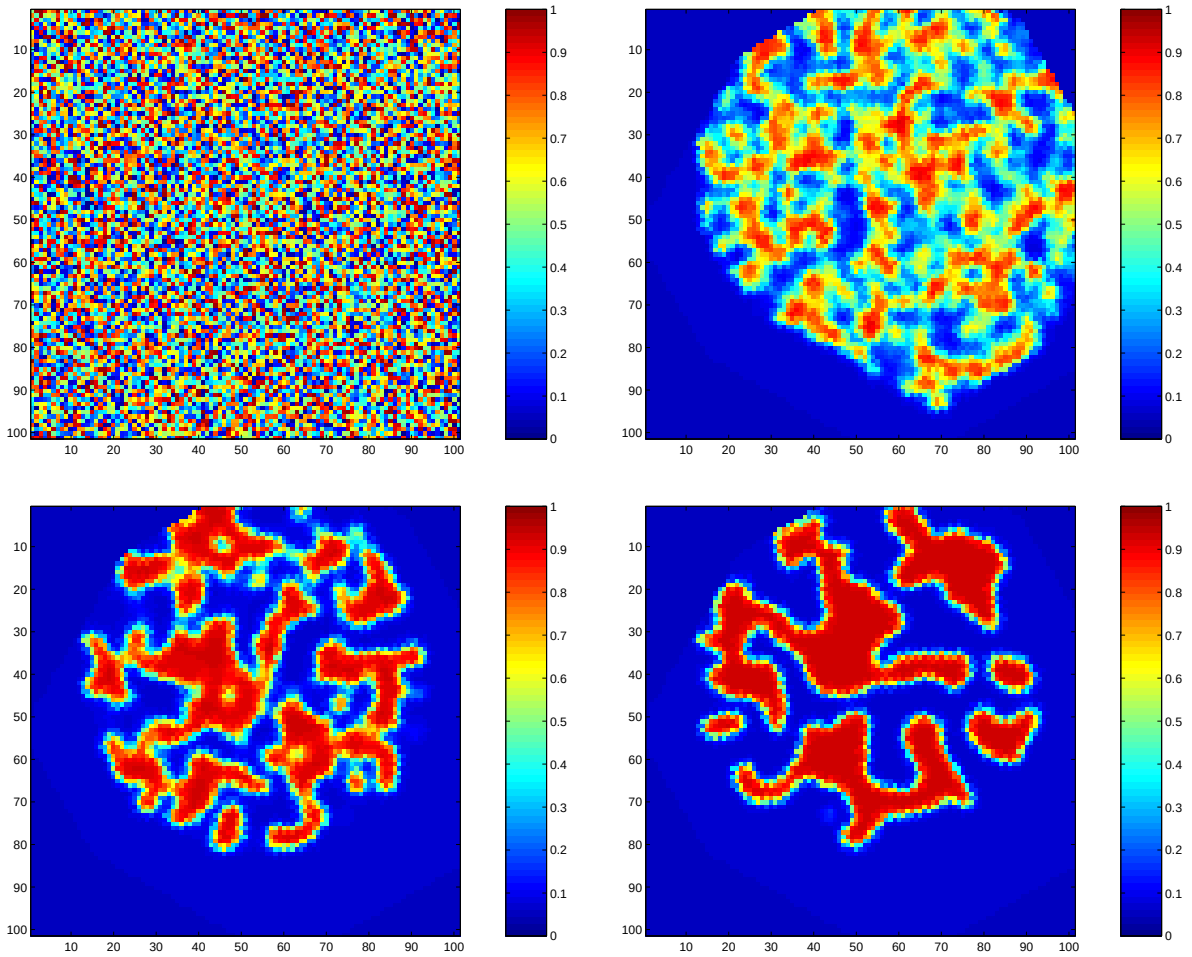

Figure 2: Sigmoid dynamics with displacement and angle shown at steps 0, 5, 10, 20.

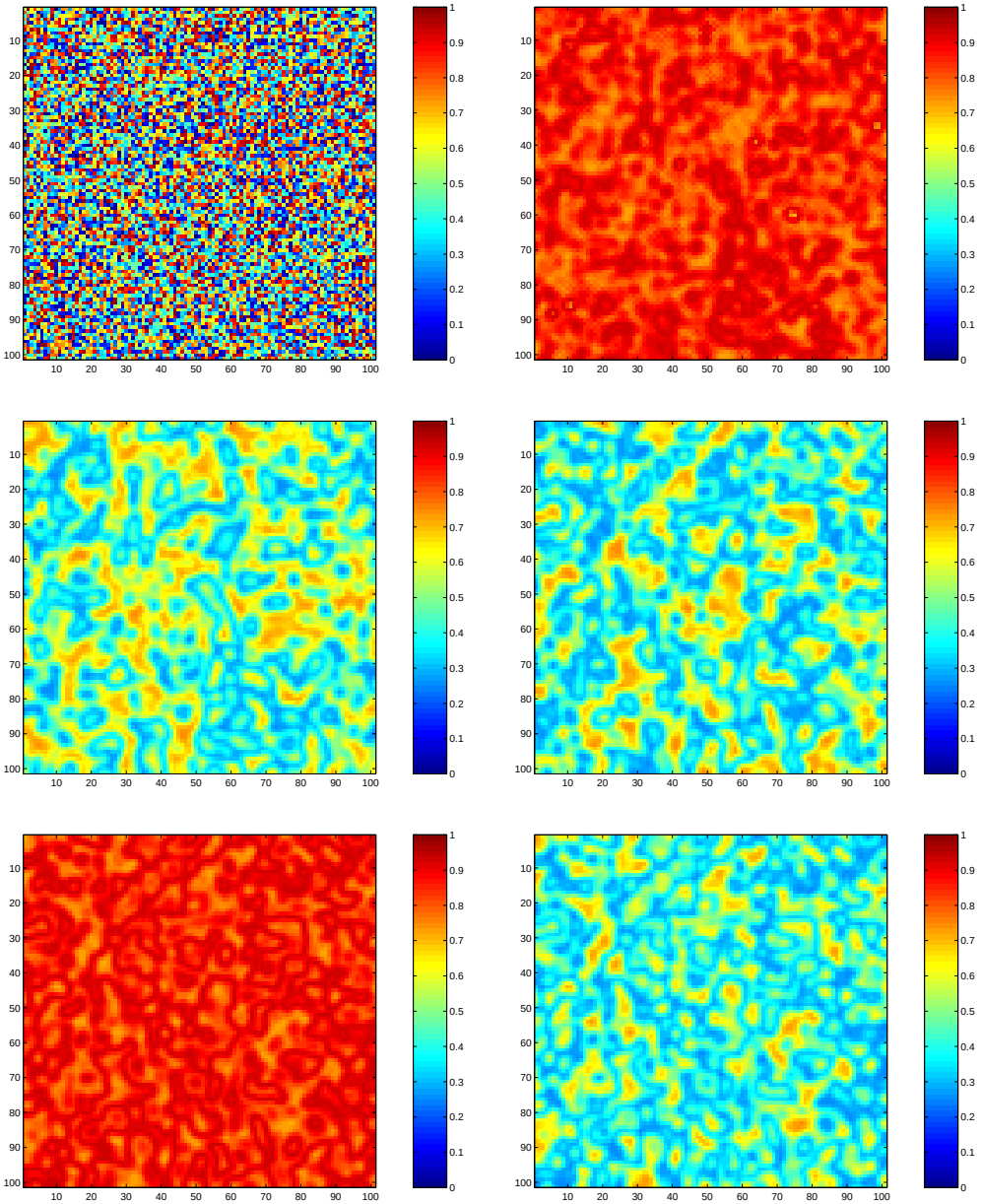

Figure 3: Logistic map dynamics shown at steps 0, 5, 10, 18, 19, 20.

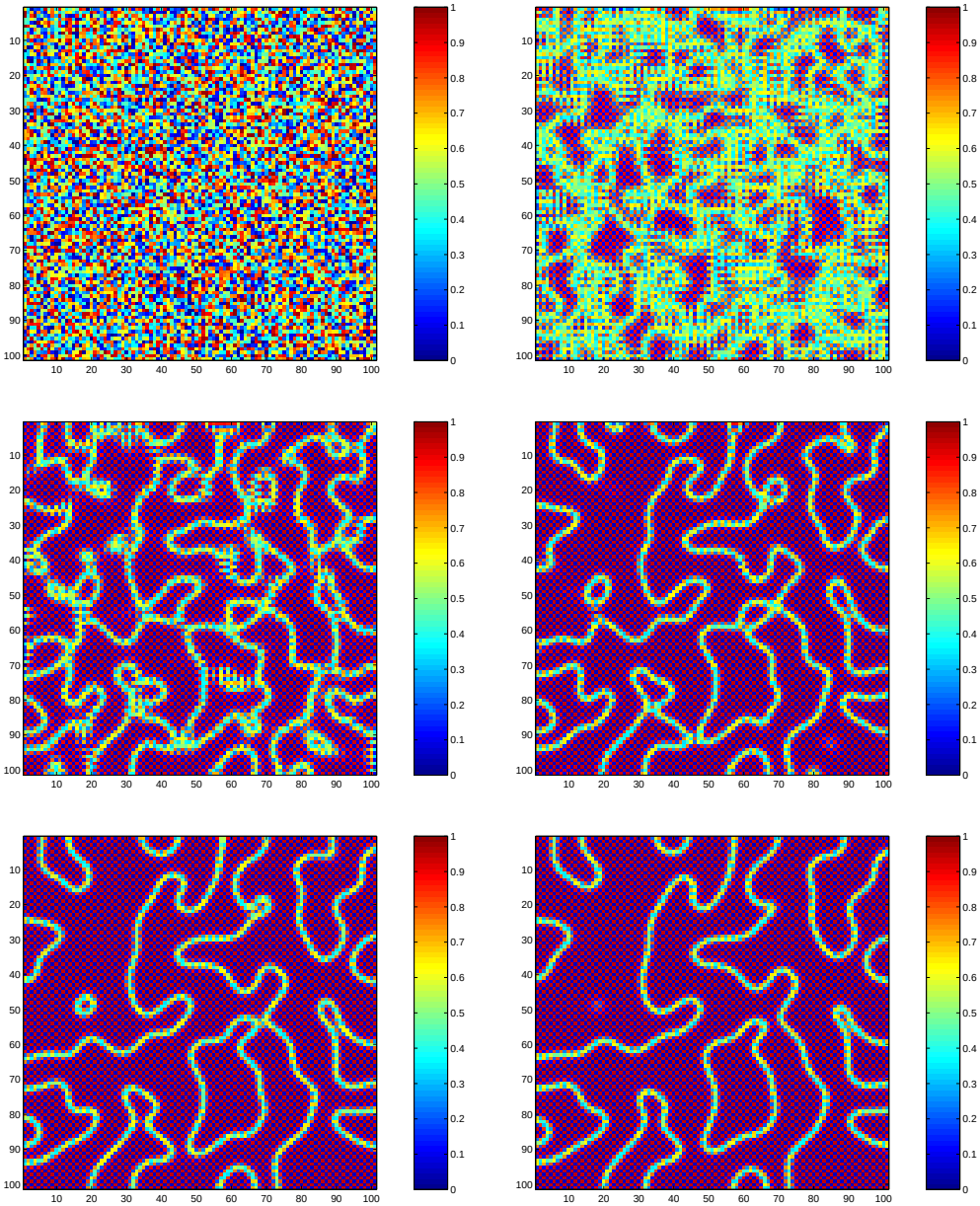

Figure 4: ON-Center cell dynamics shown at steps 0, 5, 10, 15, 20, 25.