2. Cellular paradigms: theory and simulation

2.1. Cellular systems

The main features of cellular systems are *regularity* and *homogeneity*. In fact a cellular system can be defined as a structured collection of identical elements called ***cells.*** The structure is given by the choice of a ***lattice***. Such lattices are 1-dimensional, 2-dimensional and, less used, 3 or more dimensional. The following are examples of common used 2-dimensional lattices:

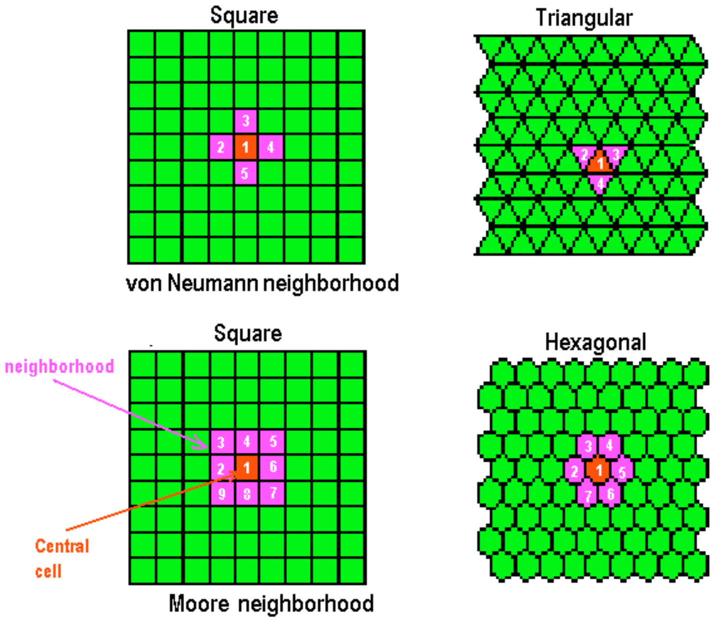

Fig.1. Cells, lattices, neighborhoods and indexes.

In the above pictures, the central cell is depicted in red while *cells in the **neighborhood*** are depicted in magenta. The *neighborhood* is another important concept which defines a cellular system and it represents the set of cells that are directly interacting with the central cell.

The basic computational unit in a cellular automata is called a *cell* and such cells are in fact ***nonlinear dynamic systems***. The cell dynamics can be ***continuous in time*** and in this case they are mathematically described by Ordinary Differential Equations ODEs or ***discrete in time*** and in such cases their dynamics is described by a *difference* equation.

<u>Example 1</u>: $y_1(t) = \sum_{k \in N} a_k \cdot y_k(t-1)$ defines the dynamics of the output $y_1(t)$ of a *discrete time **autonomous*** cell defined by a weighted summation of all neighboring cell outputs at the previous time clock. The neighborhood is represented by the set N and a **unique index** k is chosen to identify the neighboring cell in a particular neighborhood. The assignment of different integer values to k is conventional and some examples are provided

in Fig.1. where such indexes are depicted in white for each type of neighborhood. The time variable *t* is here an *integer*.

The above cell is one of the simplest possible. It is autonomous since there is no external input to drive the dynamics of the cell. In the most general case a cell is described by the following variables:

Inputs – usually denoted by variable *u* (scalar) or **u** (vector of inputs);
States – usually denoted by variable *x* (scalar) or **x** (vector of states);
Initial states- a particular state variable at the initial moment, *t=0*.
Outputs – usually denoted by variable *y* (scalar) or **y** (vector of outputs).

In terms of a computing machine, *inputs* and *initial states* are used to supply the cellular system with the *input data* (to be processed) while the result or *output data* is available in the form of a pattern (spatial, temporal or spatio-temporal) of the *outputs.* For simpler cells sometimes outputs and states may coincide, otherwise the outputs represent a nonlinear function applied to states.

<u>Example 2 :</u> **The "Standard Cellular Neural Network" cell**

The ODE defining the standard cellular neural network [Chua and Yang, 1988] is:

$$\dot{x}_1 = -x_1 + \sum_{k \in N} a_k y_k + \sum_{k \in N} b_k u_k + z\,, \qquad (1)$$

where $y_k = \frac{1}{2}\left(|x_k + 1| - |x_k - 1|\right)$,

and the initial states $x_k(0)$ are specified for all cells in the neighborhood *N*.

Observe that this is a more complex, *continuous time*, dynamic system with a nonlinearity induced by the relationship between states and outputs.

In general (non-standard CNN cell) equation (1) can be replaced with:

$$\dot{x}_1 = -x_1 + f(G, y_k, u_k) \qquad (1')$$

where *f* is a nonlinear function and *G* is a gene as defined below.

Genes

Observe that in all previous examples the dynamics of the cell for the same input excitation and the same initial state is significantly influenced by the values given to certain parameters (denoted a_k, b_k, and *z* in the above example). In [Chua, 98] it was proposed to pack all these parameters in a unique vector $\mathbf{G} = [a_1, a_2,..a_{n1},..,b_1,b_2,..b_{n2},...]$ called a ***gene*** since it determines the overall function of the cellular system much like the DNA – based gene determines the functionality of the biological systems made of cells containing that DNA.

Discrete and Continuous states / outputs

In defining a cellular system one has to define the variation domain of the state and output variables. For example, one can use ***continuous state*** cells where the outputs are defined within a bounded interval or one can also use ***discrete state*** cells where the states or/and the outputs belong to a finite set of possible values.

A **binary output** cell implements Boolean functions, i.e. it provides a logical "TRUE" or "FALSE" output for each of the 2^n possible combinations of inputs. Each input also represents a truth value. The cell can be again specified as a **discrete-time dynamical system** but it can be also specified using a **transition table** or a set of **local rules**. The last two modes of specifying a cell are specific to the Cellular Automata formalism [Toffoli & Margolus, 1987]. As we will show in the next chapter, a compact piecewise linear description (i.e. a discrete dynamical system) can be found for any Boolean or other type of input function:

Example 3: **A Boolean cell**

$$y_1(t) = sign\left(z_0 + \sum_{k \in N} b_k \cdot u_k(t) \right) \qquad (2)$$

Assuming that the set $\{-1,1\}$ is used to code the truth set $\{FALSE, TRUE\}$, the cell equation (2) above defines a family of Boolean functions. For a particular Boolean function one should specify the gene parameters. For example, the AND function with 3 inputs is associated with the following gene: $\mathbf{G} = [z_0, b_1, b_2, b_3] = [-2,1,1,1]$. If the gene is replaced with $\mathbf{G} = [z_0, b_1, b_2, b_3] = [2,1,1,1]$, the OR function with 3 inputs is implemented. The Boolean cells are studied extensively in Chapter 3, while in Chapter 5 it is shown how the emergent dynamics in a CNN made of such cells can be related to the gene parameters of the cell.

Boundary conditions

While most of the cells in a cellular automata are connected with all their neighbors on a given lattice, the cells located on the boundary of a lattice have a special regime. One should define the way in which these cells interact with their neighbors. This is called **a boundary condition**. The choice of the boundary condition may have a great influence on the cellular system dynamics, therefore this is an issue that has to be clearly specified when dealing with cellular systems. Two solutions of this problem are common:

Periodic boundary conditions: Opposite borders of the lattice are "sticked together". A one dimensional "line" becomes following that way a circle, a two dimensional lattice becomes a torus.

Reflective boundary conditions: The border cells are mirrored: the consequence are symmetric border properties.

In the remainder of the book exclusively square lattices and periodic boundary conditions are used.

2.2. Major cellular systems paradigms

The cellular automata (CA)

Stanislas Ulam proposed in the forties the first cellular system. This model, called a cellular automaton (CA) was much related with the works of Von Neumann dealing with self-reproduction and artificial life. Von Neumann asked what kind of logical organisation of an automaton is sufficient to produce self-organisation. The result of his theoretical deduction is that an universal Turing machine embedded in a cellular array using cells with 29 states and a neighborhood with 5 cells is existing [Neumann von, John, 1966]. This machine (in fact a configuration of states) is called a universal constructor, and is capable to construct any machine described on the input tape and reproduce the input tape and construction machinery. Arthur W.Burks [Burks, 1968] and E.F.Codd [Codd, 1968] demonstrated the possibility to reduce the complexity of von Neumanns automaton. They introduced a machine requiring only 8-states per cell capable of self-reproduction. Later on, the idea of using Cellular Automata to study self-reproduction was developed by Langton [Langton, 1984] focusing on rather simpler machines called loops which are capable to self-reproduce in less complicated cellular systems. Much simpler models of self-reproducing loops were proposed recently. Several cellular models producing evolvable self-reproducing loops were also proposed recently (e.g. [Sayama, 2000]).

Another well known model of CA is the Conway's "Game of Life". The game became well known after an article published in 1970 [Gardner, 1970]. It is a binary, discrete time cellular automata where the cell is defined by some simple, common sense, **local rules.** It is assumed that each cell has 8 neighbors (Moore neighborhood) and each cell is either DEAD (coded as state 0) or ALIVE (coded as state 1). There are two possibilities:

1. The cell is DEAD. Then it become ALIVE in the next cycle only if 3 of its neighbors are ALIVE. Otherwise it stays DEAD (This rule somehow suggests that a new being is generated if enough people are there around. Though, too many people may compete for resources and there is no place for a "new life").
2. The cell is ALIVE. Then except if it has 2 or 3 ALIVE neighbors it will become DEAD in the next cycle. Otherwise it will stay ALIVE. This rule suggests that a cell could die either by loneliness (only 1 living cell around) or by overpopulation (more than 3 cells around competing for resources makes life impossible).

The dynamics of such CAs is surprisingly complex and intriguing and it was extensively studied [Gardner, 1983].

At http://www.bitstorm.org/gameoflife/ the reader may find an easy-to-use, platform independent example of running the Game of Life. Many other computer simulations of this popular cellular automaton are available on the Internet.

Game of Life is a serious game. It was proved [Berlekamp *et al.*, 1982] that for certain configurations of initial states the "game of life" CA embeds a universal Turing machine and therefore, in principle, a CA with "game of life" cells is capable of universal computation. We will show later in Chapter 3 that a simple nonlinear model of the "game of life" cell exists and in Chapter 5 we show that an even wider range of emergent behaviors can be traced in cellular automata or generalized cellular automata using mutations of the "game of life" cell.

The Cellular Automata are widely studied today as a convenient paradigm for modeling physical processes and for investigating emergence and complexity. Several good on-line tutorials are available, for example [Schatten A., 1999], [Rennard, 2000], [Weimar, 1996] to name just a few of the most recent.

The Cellular Neural Network (CNN) model

The Cellular Neural Network (CNN) model was proposed by Chua [Chua & Yang, 1988] as a practical circuit alternative to Hopfield and other type of recurrent networks. The CNN cell is a **continuous time and continuous state** dynamical system with some saturated nonlinearity (see equation (1)) which is well suited for implementation using analog circuits. Unlike CAs which are mostly used to prove various theories or to model physical processes the CNN was intended from the beginning to be also a useful signal processing paradigm.

An important step towards making this paradigm an application oriented one was the introduction in 1993 of the concept of CNN Universal Machine (CNN-UM) [Roska & Chua, 1993]. Within the framework of the CNN-UM a CNN kernel is employed to perform sequentially various information processing tasks. Each task is associated with a gene from a continuously growing library of more than 200 different primitive genes (and tasks). Therefore one may combine various such primitives which are stored in an analog memory much like the instruction-code memory of digital microprocessor and combined in various ways to provide complex and sophisticated applications at TerraOps computing speed. Recent electronic implementations of the CNN-UM are in fact *sensor computers* [Roska, 2000], having the capability to sense and to process an image on the same chip.

Several generations of microelectronic chips were reported so far [Roska & Rodriguez-Vazquez, 2000a], as well as development tools which allow an user to program the CNN as visual microprocessor. There is a wide range of applications, mostly in the area of image processing. Such application include image segmentation, image compression, fast halftoning, contour tracking, image fusion, pattern recognition, to name just a few.

Although initially the equilibrium dynamics of CNNs was mostly exploited for applications, recently the non-equilibrium dynamics is employed for certain interesting applications in what is currently called “computing with waves” [Roska, 2001],[Roska, 2002].

There is a lot of research around the world in the field cellular neural network most of which is reported in the Proceedings of the IEEE CNNA workshops (Cellular Neural Networks and their Applications), ISCAS or IJCNN conferences as well as in numerous journal papers or books. Some recent tutorials about CNNs are [Chua & Roska, 2001], [Chua, 1998], [Roska & A. Rodríguez-Vásquez, 2000b], [Hänggi & Moschytz, 2000].

Several CNN simulators are freely available and the reader may check [Hänggi, 1998], and [Hänggi et al., 1999] for an easy to use, computing platform independent CNN simulator. A wide range of simulators as well as news about the progress in the CNN research can be accessed from [AnalogicLAB, 2002]. The SCNN simulator from the Applied Physics Department can be used on Unix platforms [SCNN Simulator, 2000].

The Generalized Cellular Automata

In [Chua, 1998], the idea of a Generalized Cellular Automata (GCA) was introduced as an extension of the CNN so that a GCA includes CAs as a special case. The main idea of the GCA is to use a CNN in a discrete-time loop.

In other words, for each period of the clock the CNN system evolves until it eventually reaches a steady output (for some CNN genes it is possible to have complex oscillatory dynamics) starting from a given initial condition and from inputs variables that are copies of the GCA neighboring cells outputs at the and of the previous time cycle.

The additional CNN loop is thus given by the discrete time equation $u_k(t) = y_k(t-1)$. There are two cases of interest:

1. When the CNN cell is *uncoupled* (i.e. all coefficients $a_k = 0$ for $k \geq 2$ in (1)) it was proved that the nonlinear dynamic system (1’) converges towards a steady state output solution and therefore aft enough period of time *T* the cell can be described as a simple nonlinear equation of the following form: $y_1(t) = F(\mathbf{G}, u_k(t))$ where *F* is a nonlinear function and *G* is a gene (i.e. tunable parameters). In this case, as detailed in Chapter 5, the GCA can emulate any CA, provided that there exist a method to map the local rules or transition table into the nonlinear function *F*. As it can be easily observed such GCA can also implement discrete-time but continuous state cellular automata.

2. When the CNN cell is *coupled*, one should first determine a set of proper genes such that the same steady state behavior occurs during the clock time *T*. This is often not a trivial task. Then, the behavior of the resulting GCA is more complex than that of a normal CA. The reason is that an emergent computation already takes place during the period of time *T* in the CNN, therefore at the discrete time moment when the output of a cell is sampled, it does not represent only the contribution of the neighboring cells as in the case of a "classic" cellular automata but rather the contribution of all CNN cells. In some sense one can say that employing a continuous time CNN during the clock time is equivalent to artificially increasing the neighborhood to the whole cellular space. This behavior may imply interesting computational consequences. At the end of Chapter 5 we suggest an application in image restoration based on this particular mode of operation of a GCA.

Reaction-Diffusion Cellular Nonlinear Networks

Reaction-Diffusion CNNs (RD-CNNs) were proposed in [Chua *et al.*, 1995] as a particular case of continuous-time autonomous[1] CNN which are space-discretized models of Partial differential Equations describing the Reaction-Diffusion physical processes.

From a circuit perspective a RD-CNN can be modeled as a collection of multiport nonlinear cells. These cells are coupled with their neighboring cells via linear resistive networks (Fig.2).

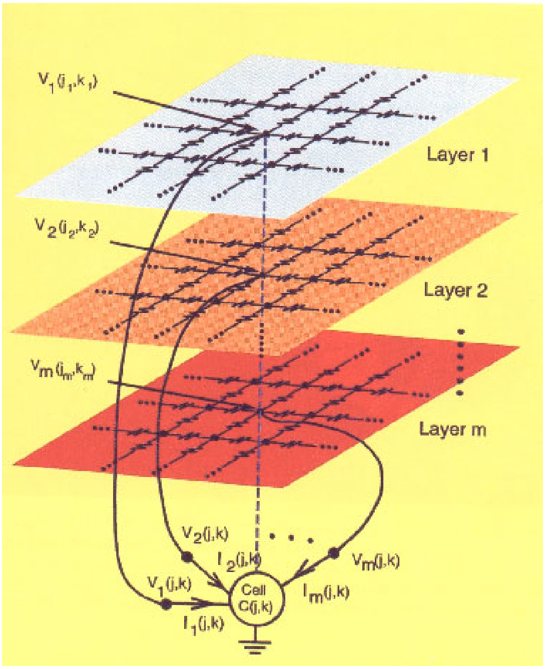

Fig.2. The topology of a Reaction Diffusion Cellular Neural Network. A cell is a *m*-port described by a nonlinear ODE which models a physical ***reaction***. The coupling with neighboring cells is done via resistive grids modeling the physical process of ***diffusion.***

[1] Autonomous means without external inputs. Information processing in such systems is based on prescribing the initial state of all cells with some pattern followed by a dynamic transformation of this pattern until a stopping criterion is met.

Example 4: **The case of a Reaction Diffusion cell with von Neumann neighborhood**.

The cell equation is of the following form:

$$\begin{aligned}
\dot{x}_1^1 &= f_1(x_1^1, x_1^2, ..., x_1^m, \mathbf{G}_1) + D_1\left(x_2^1 + x_3^1 + x_4^1 + x_5^1 - 4x_1^1\right), \\
&.. \\
\dot{x}_1^j &= f_1(x_1^1, x_1^2, ..., x_1^m, \mathbf{G}_j) + D_j\left(x_2^j + x_3^j + x_4^j + x_5^j - 4x_1^j\right), \qquad (3) \\
&.. \\
\dot{x}_1^m &= f_1(x_1^1, x_1^2, ..., x_1^m, \mathbf{G}_m) + D_m\left(x_2^m + x_3^m + x_4^m + x_5^m - 4x_1^m\right)
\end{aligned}$$

where x_k^j is the state variable of cell k (using the neighborhood index as in Fig.1 with the central cell indexed as k=1) in layer j. The gene is also distributed among layers as $\mathbf{G_1,...G}_m$ and in addition, each layer is characterized by the *scalar diffusion coefficient* D_j which corresponds physically to the conductance of the resistors in the resistive grid in Fig.2.

RD-CNNs will be studied extensively in Chapter 4 since they have the interesting property that emergent behaviors can be predicted by carrying out a *local activity* analysis of the isolated cell (i.e. ignoring the coupling). Their simulation is discussed in Section 2.4. in this chapter.

2.3. Matlab simulation of generalized cellular automata

It is convenient to simulate various types of cellular system using the programming environment Matlab (http://www.mathworks.com/products/matlab/) produced by Mathworks. We will assume here the reader has some basic knowledge of Matlab. For readers unfamiliar with this language, a good starting point might be the primers [Sigmon and Davis, 2001] or [Sigmon, 1993].

Uncoupled GCAs

In order to simulate an uncoupled GCA one should first write the function GCA_U_CELL.M which implements the GCA cell. It is in fact a simple description of the nonlinear function $y_1(t) = F(\mathbf{G}, u_k(t))$:

```
function y=gca_u_cell(u1,u2,u3,u4,u5,u6,u7,u8,u9)
Z=[-1,-2,-4,-8,-7];
B=[1 1 1 1 1 1 1 1 1];
s=-1;

sigm=B(1)*u1+B(2)*u2+B(3)*u3+B(4)*u4+B(5)*u5+B(6)*u6+B(7)*u7+B(8)*u8+B(9)*u9;
w=Z(1)+abs(Z(2)+abs(Z(3)+abs(Z(4)+abs(Z(5)+sigm))));

y=s*sign(w);
%y=0.5*(abs(w+1)-abs(w-1));
%y=w;
```

The nine inputs of this function can be either a scalar, a vector or an array. The program above implements the Parity9 local logic function. However, as seen in Chapter 3 by simply changing the gene parameters (in our case the values of the Z,B,s) many other local Boolean functions can be implemented. Moreover, one can simply change the cell structure and use any other nonlinear function. For example, as seen in the last lines, one can employ a sign function and the result is a GCA which emulates a binary cellular automaton but also continuous valued functions with or without saturation, resulting in a GCA with continuous states.

Let now have a look at the code GCA_U.M to implement the GCA itself. It is implemented in the form of a Matlab function with two input parameters. The first is a number and represents the number of iteration steps until stop (a faster stop can be achieved by pressing the keys CTRL and C). The second can be missing and if not missing it represents the name (a string) of a file containing the initial state. The initial state gives also information about the size (number of cells) of the GCA. A periodic boundary condition is implemented, the lattice is square with a Moore neighborhood indexed as explained in the source code:

```
function gca_u(steps,init_cond)
% e.g. steps=100 (runs the GCA for 100 iterations)
% e.g. init_cond='cross199'
% The inital state should be previously saved as matrix x0. For example:
% x0=ones(199,199); x0(90:110,90:110)=1
% save one199 x0
% The neighborhood index is chosen as follows:
%     9  8  7
%     6  5  4
%     3  2  1
if nargin<2
                    % by default the "cross199" initial state is loaded if no input file is specified
  init_cond='cross199';
  x0=-ones(199,199); x0(90:110,90:110)=1;
else
  eval(['load ',init_cond]);              % load the initial state
end
[m n]=size(x0);
i=1:m; % row index
j=1:n; % column index
left_j=[n,1:n-1]; right_j=[2:n,1];        % indexes of cells on the left and right
up_i=[m,1:m-1]; low_i=[2:m,1];            % indexes of cells upper and lower

y=x0;                                     % current output is loaded with x0

for s=1:steps
                                          % Computes the inputs in the next step
u9=y(up_i,left_j);    u8=y(up_i,j);        u7=y(up_i,right_j);
u6=y(i,left_j);       u5=y;                u4=y(i,right_j);
u3=y(low_i,left_j);   u2=y(low_i,j);      u1=y(low_i,right_j);
                                          % Compute the new output using the cell function
y=gca_u_cell(u1,u2,u3,u4,u5,u6,u7,u8,u9);
                                          % display the new output
image(20*y+32); axis image; colormap jet
title(['Step: ',num2str(s),' Initial state: ',init_cond,' PRESS ANY KEY TO CONTINUE']);
                                          % wait for the next step
waitforbuttonpress
end
```

With these two simple programs you may begin to observe emergent patterns in generalized cellular automata. First, running the program as they are (in fact you should run GCA_U(100) for a 100 steps run) a sequence of growing binary patterns is observed. The color code used here associates color red with +1 and blue with –1.

One can easily change the behavior by re-editing the GCA_U_CELL.M file. For example after replacing Z=[-1,-2,-4,-8,-7] with Z=[-1,-2,-4,-6,-7] the result will be a sequence of "snow-flake" like binary patterns. Moreover, if you use the initial Z but replace the initial B with B=[1 1 1 1 -8.5 1 1 1 1] and use the output function y=0.5*(abs(w+1)-abs(w-1)) the result is a sequence of colored patterns corresponding to a continuous-state cellular automaton. Observe that much of the flexibility in programming different behaviors using the same code is due to the use of a universal piecewise-linear parametric functional (the function with nested absolute values). More issues regarding cell universality are discussed in detail in Chapter 3.

Coupled GCAs

In the implementation of the cell for the uncoupled GCA there is no temporal dynamics involved and the output is just a nonlinear function of the 9 inputs. In fact, when called from the main function GCA_U the function GCA_U_CELL updates *all* cells simultaneously since it performs matrix computation (all inputs, for example u1, are matrices of the same size as the CNN). We can view GCA_U_CELL as a layer of CNN cells, but since the model is simplified there is no temporal dynamics included. In order to implement a coupled GCA we should change the cell function such that it will implement the dynamic evolution of a continuous time CNN. The resulting code of the program GCA_C_CELL.M is listed next:

```
function y=gca_c_cell(u1,u2,u3,u4,u5,u6,u7,u8,u9)
Delta_T=0.1;                   % integration step (the smallest the best precision but takes longer)
T=4;                           % the duration of the Euler integration;
B=[0 -1 0 0 1 0 0 -1 0];       % cell parameters (gene)
A=[1 1 1 1  -3 1 1 1 1];
z=0;
%-----------------------------------------------------------------------------------------------
[m n]=size(u1);
i=1:m;                         % row index
j=1:n;                         % column index
left_j=[n,1:n-1]; right_j=[2:n,1]; % indexes of cells on the left and right
up_i=[m,1:m-1]; low_i=[2:m,1]; % indexes of cells upper and lower
x=0.0001*(rand(m,n)-0.5);      % initialization of the initial state with small random values
% LINE A                       % the feed-forward contribution
ffwd=z+B(1)*u1+B(2)*u2+B(3)*u3+B(4)*u4+B(5)*u5+B(6)*u6+B(7)*u7+B(8)*u8+B(9)*u9;
for s=1:round(T/Delta_T)
  y=0.5*(abs(x+1)-abs(x-1));                  % compute the outputs for all neighbors
  y9=y(up_i,left_j);  y8=y(up_i,j);    y7=y(up_i,right_j);
  y6=y(i,left_j);     y5=y;            y4=y(i,right_j);
  y3=y(low_i,left_j); y2=y(low_i,j);   y1=y(low_i,right_j);
 % LINE B                                     % the recurrent contribution
  recur=A(1)*y1+A(2)*y2+A(3)*y3+A(4)*y4+A(5)*y5+A(6)*y6+A(7)*y7+A(8)*y8+A(9)*y9;
  x=(1-Delta_T)*x+Delta_T*(ffwd+recur);       % update the state dynamics following equation (1)
                                              %display the new output
image(20*y+32); axis image; colormap jet
title(['Time: ',num2str(s*Delta_T),' PRESS ANY KEY TO CONTINUE']);
% wait for next step
waitforbuttonpress
end
```

This new cell is in fact a implementation of the standard (linear coupling) CNN model (of Chua and Yang [Chua & Yang, 1988] but in fact can be easily developed into a nonstandard model by simply changing LINE A and LINE B accordingly. The continuous time dynamics associated to Equation (1) is emulated on the discrete-time computer using a simple integration method; namely, the Euler's method. Therefore, in addition to gene parameters this new cell has to specify two dynamic parameters. The first, **Delta_T** indicates the step size and the smallest the better will be the approximation of the continuous time dynamics. The second, **T**, represents the integration time period. Usually the user may choose such a value that corresponds to a steady state dynamics in the output.

In order to simulate a coupled GCA one should use the above file in conjunction with a file called GCA_C.M which is simply obtained from a copy of the GCA_U.M file after changing the name of cell function from GCA_U_CELL to GCA_C_CELL.

Using the values in the file above, the dynamics of the CNN will be first observed, and from time to time a figure will show the GCA output. The visualization of the CNN steps is useful for first experiments and it allows one to choose the two dynamic parameters. However, if one wants to see only the GCA output, the visualization lines in the above file should be ignored (by inserting % in front of them).

As in the previous cases, a wide range of dynamic phenomena can be simulated by re-editing the above file after proper changes of the gene or dynamics parameters.

Simulation of standard cellular neural networks

As we already discussed, simulating the CNN reduces to the particular case of simulating a coupled GCA for only one discrete time step. The GCA_C_CELL is activated and it will simulate the CNN. Its parameters and even novel CNN models can be easily simulated by properly re-editing the GCA_C_CELL.M file.

2.4. Simulation of Reaction-Diffusion Cellular Neural Networks

We will exemplify the simulation for the particular case of FitzHugh Nagumo cells (See Chapter 4 for details). However, any Reaction-Diffusion system can be easily simulated by after several minor re-editing operations on the files. As in the previous cases we should first define the cell equation. In the case of the Reaction-Diffusion systems this is done by the following function (RD_CELL.M):

```
function y=rd_cell(t,x)
% Cell gene parameters - they can be further edited (here they are specific to the FitzHughNagumo cell)
a=0.1; b=1.4; e=-0.1; alfa=1; d1=.1; d2=.5;
%-------------------------------------------------------------
[sz dm]=size(x); N=sqrt(sz/2);                    % detects the CNN size
i=1:N; j=1:N;                                     % row and column index
left_j=[N,1:N-1]; right_j=[2:N,1];                % indexes of cells on the left and right
up_i=left_j; low_i=right_j;                       % indexes of the upper and lower cells

u=reshape(x(1:N^2),N,N)'; v=reshape(x(N^2+1:2*N^2),N,N)'; %repack the two state variables
u_up=u(up_i,j); u_dwn=u(low_i,j); u_lft=u(i,left_j); u_rt=u(i,right_j);
v_up=v(up_i,j); v_dwn=v(low_i,j); v_lft=v(i,left_j); v_rt=v(i,right_j);

% Implement the cell defining equations. The functions  f1 and f2 are implemented separately for clarity
u_out=f1(u,v,alfa)+d1*(u_up+u_dwn+u_lft+u_rt-4*u);
v_out=f2(u,v,a,b,e)+d2*(v_up+v_dwn+v_lft+v_rt-4*v);;
y=[reshape(u_out',1,N^2), reshape(v_out',1,N^2)]'; %repack the two state variables
```

Here a two-layer Reaction-Diffusion CNN is implemented. The Reaction-Diffusion systems in Chapter 4 are all 2-layers systems. The function RD_CELL.M is prepared to be used in an Matlab specific integration routine based on the Runge-Kutta algorithm. Since the integration routine ODE23 requires a unique vector variable **x** , before preparing to write down the cell equations as in (3) we have first to unpack the vector variable into the two variables **u** (first layer) and **v** (second layer). The two variables are square matrices of dimension N each while an element u(i,j) of each matrix corresponds to the CNN cell on the (i,j) position.

The first line allows one to specify the gene parameters of the cell. Different dynamic behaviors can be simulated by changing these values and saving the file. The next lines deal with the implementation of (3). Note that the nonlinear functions f_1 and f_2 in (3) are implemented separately being specific for the FitzHugh-Nagumo cell model. They are as follows:

```
function out=f1(x,y,alfa)
out=-y+alfa*x-(1/3)*(x.^3);
```

```
function out=fhn2(x,y,a,b,e)
out=-e*(x-b*y+a);
```

Whenever a different type of cell (of order two) has to be implemented one needs only to change the above function and to provide the adequate list of parameters in RD_CELL.M. No other change is required in the file. Next let us examine the source code of the main simulator function called RD_CNN.M

```
function rd_cnn(mode,init_state)
% mode=0 -> displays 5 snapshots only, mode=1 -> displays 42 snapshots
ts=0:1:40; % Dynamic evolution (here time from 0 to 40 with snapshots from 1 to 1 time units)
%--------------------------------------------------------------------
if nargin<2 u=(rand(20,20)-.5)*0.1; v=(rand(20,20)-.5)*0.1;
else eval(['load ',init_state]); end
% If there is only one parameter random initial states are generated, otherwise loaded from a file
[N N]=size(u);                                          %Evaluate the CNN dimension (square CNN only)
if mode==1 n_snp=42; elseif mode==0 n_snp=5; end; %The number of snapshots for display
% LINE A - THE ODE INTEGRATION
y0=[reshape(u',1,N^2), reshape(v',1,N^2)];              % pack the two state variables
opt=odeset('OutputSel',ts);                             % initialize the intgration routine
[t out]=ode23('rd_cell',ts,y0,opt);                     % integrate the rd_cell using ode3

u=out(:,1:N^2); v=out(:,N^2+1:2*N^2);                    %unpack the state variables
mi=min(min(u)); ma=max(max(u)); mi_v=min(min(v)); ma_v=max(max(v)); [per dum]=size(t);
disp(['u_max: ',num2str(ma),'  u_min: ',num2str(mi)]);

% DISPLAY RESULTS
if mode==0 set(figure(1),'Position', [263 530 770 175]);
elseif mode==1 set(figure(1),'Position',[172 95 862 641]); end
colormap jet; whitebg(1, [1 1 .5]); ti=[]; col=6; linii=ceil(n_snp/col);
for i=1:n_snp
        tsh=round(1+(per-1)*(i-1)/n_snp);
        ti=[ti t(tsh)];
        im=reshape(u(tsh,:),N,N)';
        eval(['img',num2str(i),'=64*(im-mi)/(ma-mi);']);
        subplot(linii,col,i); image(64*(im-mi)/(ma-mi)); axis image; axis off; title(['t=',num2str(t(tsh))]);
end
```

This program allows the integration of the ODE description of the cell in file RD_CELL.M and it allows also to display the dynamic evolution as a sequence of 5 or 42 snapshots depending on the selection of the input variable MODE. The method of integration is the one specific of the function ODE23, but one can simply replace it with other ODE integration routines available in Matlab such as ODE45, ODE113, etc.

The only parameter which has to be specified is the vector **ts** which specifies the moments of time for display, e.g. ts=1:5:67 means that there will be displayed snapshots taken at the moments 1,6,11,..67 i.e. from 5 to 5 time periods.

Each snapshot is a colored image and the color of each pixel codes the amplitude of the state variable *u* using the colormap code assigned to the display. The colormap code used by default is JET but it can be easily changed with one of the other available colormaps . During the simulation, the maximum and minimum values of the state variables are recorded and typed to the screen. When the snapshots are displayed these extreme values are assigned to the colors representing the minimum and respectively the maximum amplitude. For JET, the color assigned to minimum is dark blue and for maximum dark red.

Examples of simulation: Using the default parameters and typing RD_CNN(1) a dynamic evolution towards a non-homogeneous stable state is observed. If the parameters are changed as follows: a=0.0; b=1.3; e=-0.1; alfa=1; d1=.1; d2=0; in RD_CELL.M and ts=0:5:200 in RD_CNN.M the result of running RD_CNN(1) is an emergence of colliding waves. As shown in Chapter 4, a precise location of the sets of parameters producing interesting and potentially useful emergent behaviors can be done employing the local activity theory.

2.5.Concluding remarks

Cellular systems are described by simple mathematical equations. As shown above, using a high level platform such as Matlab, all major cellular systems paradigms can be simulated using very compact code. Although their description is compact, a wide palette of emergent behaviors can be observed and the main problem is how to choose such parameters to obtain computationally useful behaviors.

The next chapters deal with this problem. First, in Chapter 3 it is shown that universal CNN cells for both binary and continuous state computation can be compactly defined using piecewise-linear models. Then, in Chapter 4 we will investigate how the theory of local activity can be employed to locate sets of parameters for which emergent behaviors occur in Reaction-Diffusion type cellular system. Emergence in generalized cellular automata is the topic of Chapter 5 while in Chapter 6 a promising application in biometric authentication of the emergent patterns in generalized cellular automata is presented.