AutomataHex Program Notes

The following is a description of some aspects of the AutomataHex program that is used for a cellular automata model of a leaf epidermis. Sections below will highlight (1) running the program, (2) the model equations for the cellular automata, (3) output files produced by the program, and (4) some aspects of the code for the program itself.

1. Running the program

The program itself is named "AutomataHex" because an earlier version considered automata cells with 4 neighbors as opposed to this version with 6 neighbors (hexagonal cells).

Starting the program shows the basic program window with the menu items at the top, a default display of the automata grid with some values at the bottom and a color bar on the left showing the color range for water flow out of the stomata, which will appear as colored dots in the grid.

Typically, one would start by selecting the dimensions of the grid ("Dimension" menu item). This is generally the starting point for creating a new model because the grid dimensions determine several memory allocation needs by the program.

Next, one can select the numbers and locations for stomata in the grid ("Stomata" menu item). If entering a new value for the number of stomata at the top of the Stomata dialog, press the "Apply" button to allocate the right amount of memory for storing stomata locations. The next fields show the x and y values for the stomata locations. Although you can enter actual values for the locations, more commonly you would use the next parts to randomly pick locations. Enter a minimum value for their spacing and a value for how close they can be located to the grid border. Then press the "Randomize" button. A small spacing like 1.0 will tend to generate stomata with nearly random locations with clustering possible. A large value will create more ordered stomatal patterns. Note that if you enter a large spacing for a large number of stomata, the location process may not be able to succeed and fit that number of stomata and will thereby quit after one million location guesses and display a "Randomize Failure" message (see program description for details here).

Next, you would select values for some of the model parameters (the "Set params" menu item; see the Model Equations section for more detail on these values). Some default values will be shown. The value shown as kepi corresponds to the model variable k_{epi} which describes the flow conductance between the epidermal cells. The value shown as kin corresponds to the model variable k_{in} which describes the flow conductance between the bulk leaf and the epidermis. The stomatal control section shows the four values that describe the sensitivity of the stomata to their water potential (*p* values). See the Model Equations section below for some possible sensitivity values.

Next, a value can be entered for the P value of the bulk leaf or alternately to use a fixed offset from the mean P value of the epidermis. Also shown is the name that will be used if you choose to save the model solution to a file ("output.dat" is the default). The "Output Files" section below describes the content of this file that will be created.

The "Load Params" menu item will read and apply the parameter set that has been saved previously in one of the output files. This would allow you to pick those same parameter values for a current model being developed.

The "Display" menu item allows you to set the range of P values that will be displayed in gray-scale in the grid and the range of color values that will be displayed for the stomata. The "Auto" check boxes will let the program determine these range values from the model results. The top two values in this dialog determine the time step to display as the solution develops and how often (in milliseconds) the display is updated (see the material on the "Run…" menu item below.

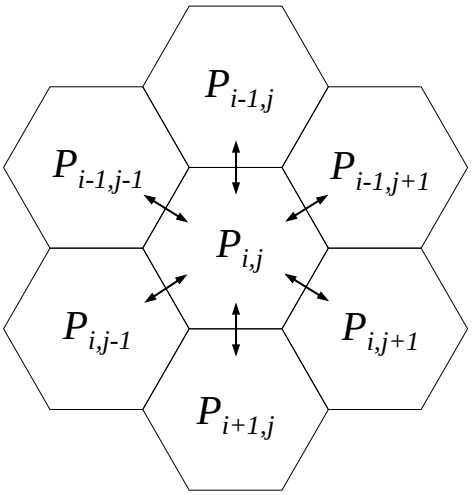
Selecting the "Run…" menu item brings up a submenu for the various ways of solving the model and displaying the resulting solutions. Here are the submenu items:

- Start – run the model continuously one step at a time using the intervals set in the Display menu item.
- Stop – stops the running of the model. Note that pushing the keyboard space bar toggles between the Start and Stop running options
- Step – advances the model one time step.
- Restart – restores the initial default values for the solution (generally P = 0 for all cells).
- Save – writes the current model results (including stomata locations and parameter values to the filename specified under "Set params".
- Start Fullspeed and Stop Fullspeed – this uses a part of the program that will run the model solution as fast as possible by only updating the display periodically (see the Program Code section for details)

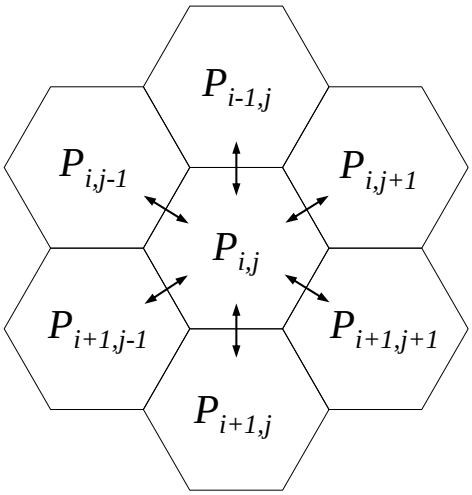
2. Model Equations

AutomataHex version - Flow with six neighbors (assumes hexagonal cells)

Odd column (j is odd)



Even column (j is even)



A) Flow from cell-to-cell in the epidermis

For *j* odd:

$$\Delta P_{i,j}^{epi} = -k_{epi}\left(6P_{i,j} - P_{i-1,j} - P_{i+1,j} - P_{i-1,j-1} - P_{i-1,j+1} - P_{i,j-1} - P_{i,j+1}\right)$$

For *j* even:

$$\Delta P_{i,j}^{epi} = -k_{epi}\left(6P_{i,j} - P_{i-1,j} - P_{i+1,j} - P_{i,j-1} - P_{i,j+1} - P_{i+1,j-1} - P_{i+1,j+1}\right)$$

B) Flow out through stomata

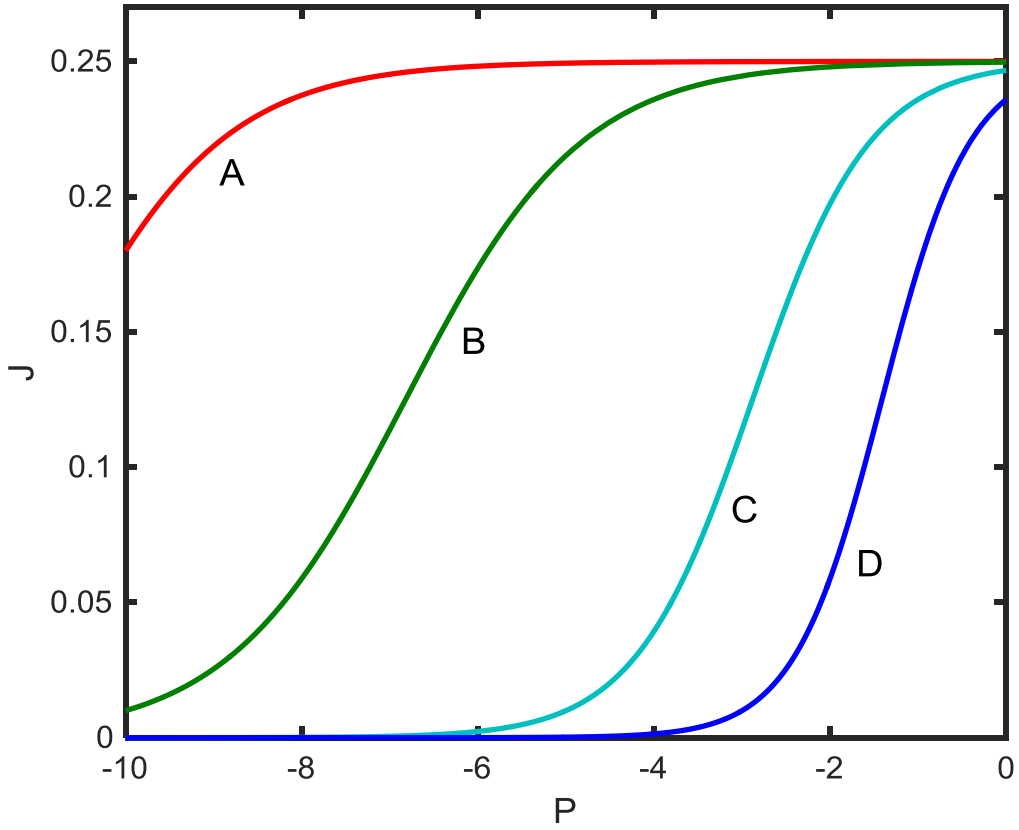
For the cells that are designated as stomata, a further adjustment (in addition to that describing flow from cell-to-cell) is made to reflect the flow out through the stomata to the atmosphere:

$$J_{out} = \frac{J_{max}\, J_0}{J_0 + \left(J_{max} - J_0\right)e^{-a\left(P - P_{Jmin}\right)}}$$

J_{max} is the maximum flow (corresponding to the most open stomata possible), P is the water status of the particular stomatal cell, J_0 is the flow if P is equal to P_{Jmin}, P_{Jmin} is the cell water status leading to a flow of J_0, and a is a stomatal sensitivity coefficient that determines how strongly flow declines with low water status.

$$\Delta P_{i,j}^{stom} = -J_{out}$$

Here are some possible examples of the stomatal control function:



The parameters for the above functions are:

Version	Sensitivity	J_{max}	J_0	a	P_{Jmin}
A	Low	0.25	0.18	1.0	-10.0
B	-	0.25	0.01	1.0	-10.0
C	-	0.25	0.01	1.5	-5.0
D	High	0.25	0.01	2.0	-3.0

C) Flow into cells from the rest of the leaf internal to the epidermis

$$\Delta P_{i,j}^{leaf} = -k_{in}\left(P_{i,j} - P_{leaf}\right)$$

The value of P_{leaf} may be either a fixed constant or an offset from the mean epidermis P value.

The total change in each cell water status is the sum of the epidermis and leaf flows plus stomatal flow, if the cell is a stoma.

D) Stomatal locations

The procedure for setting the stomatal locations is partly random but with a supplied minimum spacing. Therefore this minimum spacing allows for a range of nearly random locations (including some closely spaced stomata) to a very ordered arrangement where stomata have a large minimum spacing.

The process basically uses the standard C random number generator to create a random number that is then scaled. First, the random number is scaled to a value between the specified border width (a zone to exclude stomata near the edge of the model) and the number of columns in the model grid. Similarly, a random number is scaled between the specified border width and the number of rows in the model grid.

The above process would create nearly random locations for the stomata. To ensure unique locations and ensure a minimum spacing (the variable named spacing) between stomata, this function then searches through the locations of all previously located stomata and calculates the distances between the new location and each previous stoma location (square root of the sum of the x and y distances squared). If this distance is less than the value of the variable spacing, the new location is rejected and the process continues. Otherwise, the new location is set for the current stomata. This process continues until the planned numbers of stomata have been located.

Under certain conditions, the above process may not be able to find unique locations for all of the stomata. This becomes more likely to occur when the spacing value gets large. To avoid the process getting trapped infinitely, it will "give up" after 1,000,000 attempts to locate stomata.

E) Dimensional aspects - Consideration of capacitance

Capacitance of the cells is not considered in this model. Doing so would make the calculations of P (cell water potential) more consistent from a dimensional perspective.

From the usual definition of capacitance (C):

$$C = \frac{dV}{d\Psi}.$$

In my model, flow into a cell (J) is

$$J = k\,\Delta\Psi,$$

where $\Delta\Psi$ is the model calculated P difference between a cell and each of its neighbors and would be the driving force for water movement between the cells. By definition,

$$J = \frac{dV}{dt},$$

giving

$$\frac{dV}{dt} = k\,\Delta\Psi .$$

From the definition of capacitance above

$$\frac{1}{C} = \frac{d\Psi}{dV},$$

and so we can say

$$\frac{d\Psi}{dV}\frac{dV}{dt} = \frac{1}{C}k\Delta\Psi ,$$

and by the chain rule

$$\frac{d\Psi}{dt} = \frac{1}{C}k\Delta\Psi .$$

Therefore the change in cell water potential over one time step would be calculated from J as in the current program divided by C. With consideration for capacitance, the value of k / C would be applied to the model parameter k_{epi}. Note that the relation

$$D = \frac{k}{C}$$

describes a diffusivity (D) as appearing in the classical diffusion PDE.

F) Spatial aspects of the model

One can ask: How big are the cells in the model? Spatial distances do not appear to be present in the model equations above. Further consideration of the coefficients (k and C) will show how the distances between cell centers are part of these coefficients.

In searching for biological meaningful values for the coefficient k, one can apply estimates of the diffusion coefficient (D) for plant tissues. A value for water storage

tissue in a succulent species is cited by Smith and Nobel (1986) as 2 *E*-10 m^2/s. As derived in Schulte and Costa (1996):

$$D = \frac{K^l}{C_l} \qquad K^l = k\,\Delta x \qquad C_l = \frac{C}{\Delta x}$$

where the subscript and superscript l refer to per unit length quantities for these parameters and Δx is a representative length: for this model, the distance between cell centers. Making substitutions gives

$$\frac{k}{C} = \frac{D}{(\Delta x)^2}$$

Using the above cited diffusion coefficient and a value of 40 μm for cell size, gives

$$\frac{k}{C} = 0.125\ \mathrm{s}^{-1}$$

This value would correspond to the parameter k_{epi} in the model.

Note that this choice for parameters provides the units for the time step of seconds.

3. Output File

Here is part of a sample data output file along with comments (after the <- characters):

```
800 800 200 200 45000      <- x dim of map (pixels), y dim of map (pixels),
                           nrows, ncols, time
1.0000e-001 1.0000e-003 <- kepi, kin
400 5 2   0.2500   0.0100   1.5000  -5.0000   0.0000  <- nstomata, border,…
120 35 0.148906992      …spacing, Jmax, J0, a, pminj, pleafoff
124 181 0.174131006     <- nstomata rows with x, y locations and Jout
46 79 0.190062886       …
67 157 0.184095948      …
…
…
/* Next are values of p starting with row 0 and all columns in that row…

 -1.2529   -1.2547   -1.2649   -1.2744   -1.2922   -1.3085   -1.3328   -1.3545   -
1.3835   -1.4079   -1.4388   -1.4619   -1.4900   -1.5060   -1.5252   -1.5287   -
1.5346   -1.5251   -1.5193   -1.5021   -1.4905   -1.4718   -1.4603   -1.4443   -
1.4364   -1.4253   -1.4226   -1.4174   -1.4205   -1.4208   -1.4292   -1.4338   -
1.4459   -1.4527   -1.4662   -1.4730   -1.4862   -1.4921   -1.5046   -1.5102   -
1.5228   -1.5293   -1.5431   -1.5510   -1.5665   -1.5756   -1.5922   -1.6019   -
1.6188   -1.6282   -1.6448   -1.6535   -1.6694   -1.6769   -1.6915   -1.6969   -
1.7090   -1.7105   -1.7180   -1.7138   -1.7155   -1.7062   -1.7034   -1.6916   -
1.6876   -1.6770   -1.6751   -1.6680   -1.6701   -1.6678   -1.6747   -1.6772   -
```

```
1.6888   -1.6952   -1.7104   -1.7191   -1.7360   -1.7447   -1.7609   -1.7676   -
1.7813   -1.7850   -1.7960   -1.7973   -1.8063   -1.8063   -1.8142   -1.8135   -
1.8209   -1.8195   -1.8263   -1.8241   -1.8299   -1.8266   -1.8312   -1.8264   -
1.8294   -1.8234   -1.8253   -1.8188   -1.8207   -1.8148   -1.8176   -1.8130   -
1.8170   -1.8133   -1.8181   -1.8149   -1.8198   -1.8162   -1.8206   -1.8160   -
1.8190   -1.8125   -1.8134   -1.8043   -1.8023   -1.7886   -1.7804   -1.7578   -
1.7397   -1.7090   -1.6852   -1.6551   -1.6346   -1.6115   -1.5982   -1.5811   -
1.5718   -1.5540   -1.5412   -1.5162   -1.4954   -1.4641   -1.4384   -1.4058   -
1.3799   -1.3492   -1.3256   -1.2982   -1.2779   -1.2544   -1.2380   -1.2192   -
1.2077   -1.1944   -1.1885   -1.1813   -1.1817   -1.1812   -1.1883   -1.1946   -
1.2085   -1.2211   -1.2410   -1.2580   -1.2814   -1.2982   -1.3193   -1.3284   -
1.3393   -1.3347   -1.3317   -1.3160   -1.3040   -1.2841   -1.2700   -1.2512   -
1.2391   -1.2241   -1.2162   -1.2062   -1.2035   -1.1993   -1.2026   -1.2049   -
1.2151   -1.2248   -1.2426   -1.2603   -1.2865   -1.3122   -1.3462   -1.3775   -
1.4157  -1.4456  -1.4792  -1.4964  -1.5141  -1.5122  -1.5114  -1.4941  -
<- Next would be the second row and so forth
1.4800  -1.4547  -1.4354  -1.4108  -1.3946  -1.3775  -1.3700  -1.3623
 -1.2576   -1.2589   -1.2702   -1.2790   -1.2983   -1.3136   -1.3403   -1.3605   -
1.3933   -1.4154   -1.4519   -1.4714   -1.5076   -1.5179   -1.5467   -1.5417   -
1.5559   -1.5368   -1.5362   -1.5114   -1.5031   -1.4793   -1.4701   -1.4507   -
1.4447   -1.4312   -1.4304   -1.4232   -1.4285   -1.4269   -1.4382   -1.4406   -
1.4566   -1.4604   -1.4785   -1.4813   -1.4991   -1.5004   -1.5170   -1.5182   -
1.5346   -1.5371   -1.5547   -1.5590   -1.5785   -1.5839   -1.6050   -1.6106   -
1.6323   -1.6373   -1.6588   -1.6628   -1.6838   -1.6866   -1.7069   -1.7072   -
1.7258   -1.7215   -1.7361   -1.7251   -1.7331   -1.7167   -1.7185   -1.7008   -
1.7003   -1.6852   -1.6862   -1.6756   -1.6806   -1.6752   -1.6852   -1.6848   -
1.7000   -1.7034   -1.7231   -1.7282   -1.7505   -1.7548   -1.7770   -1.7782   -
1.7979   -1.7956   -1.8122   -1.8077   -1.8219   -1.8164   -1.8295   -1.8236   -
1.8362   -1.8297   -1.8418   -1.8344   -1.8458   -1.8371   -1.8472   -1.8368   -
1.8453   -1.8336   -1.8405   -1.8286   -1.8350   -1.8243   -1.8317   -1.8225   -
1.8313   -1.8230   -1.8329   -1.8248   -1.8350   -1.8264   -1.8364   -1.8264   -
1.8354  -1.823
```

Reading the P grid data from the output file with MATLAB

The saved data file from the program contains parameter values, stomata locations and corresponding values of J, along with all of the values in the P matrix at the current time step. One way to examine the P values for individual locations would be to import the data into MATLAB.

a) Edit the output data file to eliminate all the data before the P values and save the file under some modified file name.

b) From the MATLAB Home tab, select "Import Data". Select the file named in the previous step.

c) After a brief time for reading the file, an import wizard will open. The only change needed will likely be to select "Numeric Matrix" for the import data type. Select "Import Selection" from the import data window.

d) The data should now appear in the MATLAB Workspace with a variable name reflecting the input file name. Double-click on that name to open the usual "spreadsheet" view of the data. For some reason, on occasion an extra column appears with values of "Nan" and that column would have to be deleted.

4. Program code notes

Note that this section uses a lower-case p for the cell state while other model descriptions use an upper-case P. This is done here for consistency with the actual program code.

Here are comments from the start of the AutomataHex.cpp file:

```
This is a program that implements a 2D cellular automata model of a leaf
epidermis.  The dependent variable being modeled is the cell water
potential (p).  Each step  considers water movement (or more accurately p
changes) as determined by:
   1) the p of neighboring cells (if p of a neighbor cell is higher that
      the cell under consideration, that cell's p increases and the
      neighbor's drops),
   2) water uptake from xylem feeding all epidermal cells, and
   3) loss to the air from cells designated as "stomata".
   4) Stomata are variable as a function of cell p

Cell p values are displayed graphically in the program window as a map
with p coded in a gray scale from black (low p) to white (high p).

The degree of stomatal opening (the value of flow out through stomata) is
indicated using color values from blue through red.

The program allows for the setting of all sorts of model, solution method,
and display parameters through dialog boxes.
```

Fundamentally, this program is a straight C-language GUI without using the MFC classes available from the MS VC++ system. See the program code near the start for a table listing all the variables used by the program. Here are some functions and their uses:

FreeMem – frees the malloced arrays for the variables p and pnew
Startup – Allocates arrays and sets some initial values
NewStomata – Reallocate memory for stomatal locations
Move - Execute one move (one time step) of the model
Randomize – Randomly select new locations for stomata
MoveFull – Execute moves as a new thread
OpenNewFile – Get a file name to open
ReadFileData – Read data from the file

The Randomize function: The procedure for setting the stomatal locations is partly random but with a supplied minimum spacing. Therefore this minimum spacing allows for a range of nearly random locations (including some closely spaced stomata) to a very ordered arrangement where stomata have a large minimum spacing.

The process basically uses the standard C random number generator to create a random number that is then scaled. In one case the random number is scaled to a value between the specified border width (a zone to exclude stomata near the edge of the model) and the number of columns in the model grid. Similarly, a random number is scaled between the specified border width and the number of rows in the model grid.

The above process would create random locations for the stomata. To ensure unique locations and ensure a minimum spacing (the variable named spacing) between stomata, this function then searches through the locations of all previously located stomata and calculates the distances between the new location and each previous stoma location (square root of the x and y distances squared). If this distance is less than the value of the variable spacing, the new location is rejected and the process continues. Otherwise, the new location is set for the current stomata. This process continues until the planned number of stomata have been located.

Under certain conditions, the above process may not be able to find unique locations for all of the stomata. This becomes more likely to occur when the spacing value gets large. To avoid the process getting trapped infinitely, it will "give up" after 1,000,000 attempts to locate stomata.

The Move function: works through the grid of cells calculating the new values of p based on the values of the 6 neighboring cells, uptake from the leaf, and loss if the cell is a stoma. Note that this calculation differs for cells on the boundaries of the grid, where other cells are missing in certain directions. Thus the calculations are carried out separately for cells on the boundaries and in the corners of the grid. The max, min, and mean values of p and $Jout$ are calculated for scaling the color or grayscale of these values in the display. This function advances the solution by one time step.

When the automata is advancing continuously, the time and display settings from the Display dialog are used to determine how often (in clock time) the move process occurs and if more than one step should occur between updates of the display.

Program function used in the AutomataHex program for each time step

```
   void Move( int t )      //Execute one "move"
   {
      int i, j, k;

      if ( bOffset ) pleaf = meanp + pleafoff;       //If checked, pleafoff is an
offset from meanp
      else   pleaf = pleafoff;   //Oterhwise pleafoff is just a constant value
      for( i=1; i<nrows-1; i++ ) //Do all interior cells
         {
            for( j=1; j<ncols-1; j++ )
```

```
                {
                if( (j % 2) == 0 )  //column is even
                    {
                    dp = -kepi * ( 6 * p[i][j] - p[i-1][j] - p[i+1][j] - p[i][j-1]
                                   - p[i][j+1] - p[i+1][j-1] - p[i+1][j+1] );
                    }
                else             //column is odd
                    {
                    dp = -kepi * ( 6 * p[i][j] - p[i-1][j] - p[i+1][j] - p[i-1][j-1]
                                   - p[i-1][j+1] - p[i][j-1] - p[i][j+1] );
                    }
                dp += kin * ( pleaf - p[i][j] );
                pnew[i][j] = p[i][j] + dp; //Temp store values in pnew
                }
            }
        j = 0;                  //Do left column, except for corners
        for( i=1; i<nrows-1; i++ )
            {
               dp = -kepi * ( 4 * p[i][j] - p[i-1][j] - p[i+1][j] - p[i][j+1] -
p[i+1][j+1] );
               dp += kin * ( pleaf - p[i][j] );
               pnew[i][j] = p[i][j] + dp;
            }
        j = ncols-1;            //Do right column, except for corners
        for( i=1; i<nrows-1; i++ )
            {
               dp = -kepi * ( 4 * p[i][j] - p[i-1][j] - p[i+1][j] - p[i][j-1] -
p[i+1][j-1] );
               dp += kin * ( pleaf - p[i][j] );
               pnew[i][j] = p[i][j] + dp;
            }
        i = 0;                  //Do top row, except for corners
        for( j=1; j<ncols-1; j++ )
            {
            if( (j % 2) == 0 )  //even column in top row
                {
                dp = -kepi * ( 5 * p[i][j] - p[i+1][j] - p[i][j-1] - p[i][j+1]
                               - p[i+1][j-1] - p[i+1][j+1] );
                }
            else             //odd column in top row
                {
                dp = -kepi * ( 3 * p[i][j] - p[i+1][j] - p[i][j-1] - p[i][j+1] );
                }
            dp += kin * ( pleaf - p[i][j] );
            pnew[i][j] = p[i][j] + dp;
            }
        i = nrows-1;            //Do bottom row, except for corners
        for( j=1; j<ncols-1; j++ )
            {
            if( (j % 2) == 0 )  //even column in bottom row
                {
                dp = -kepi * ( 3 * p[i][j] - p[i-1][j] - p[i-1][j-1] - p[i-1][j+1] );
                }
            else             //odd column in bottom row
                {
                dp = -kepi * ( 5 * p[i][j] - p[i-1][j] - p[i-1][j-1] - p[i-1][j+1]
                               - p[i][j-1] - p[i][j+1] );
                }
            dp += kin * ( pleaf - p[i][j] );
            pnew[i][j] = p[i][j] + dp;
            }
                                    //Upper-left corner
        pnew[0][0] = p[0][0] - kepi * ( 3 * p[0][0] - p[0][1] - p[1][1] - p[1][0] ) +
```

```
                    kin * ( pleaf - p[0][0] );
                                        //Upper-right corner
        pnew[0][ncols-1] = p[0][ncols-1] - kepi * ( 2 * p[0][ncols-1] - p[1][ncols-1]
                         - p[0][ncols-2] ) + kin * ( pleaf - p[0][ncols-1] );
                                        //Lower-left corner
        pnew[nrows-1][0] = p[nrows-1][0] - kepi * ( 2 * p[nrows-1][0] - p[nrows-2][0]
                         - p[nrows-1][1] ) + kin * ( pleaf - p[nrows-1][0] );
                                        //Lower-right corner
        pnew[nrows-1][ncols-1] = p[nrows-1][ncols-1] - kepi * ( 3 * p[nrows-1][ncols-1]
                           - p[nrows-1][ncols-2] - p[nrows-2][ncols-2]
                           - p[nrows-2][ncols-1] )
                           + kin * ( pleaf - p[nrows-1][ncols-1] );

        if( bStom ) {
            maxJ = 0;     //If auto is checked for stomata, want min and max J
            minJ = Jmax; }
        for( k=0; k<nstomata; k++ ) {
            Jout[k] = Jmax * J0 / ( J0 + ( Jmax - J0 ) //logistic control function
                      * exp( -a * ( p[stomY[k]][stomX[k]] - pminj ) ) );
            //Use the next line to only subtract off the stomatal trans
            pnew[stomY[k]][stomX[k]] -= Jout[k];
            //Use the next line to subtract off both the stomatal trans and inflow from
the leaf
    //      pnew[stomY[k]][stomX[k]] -= ( Jout[k] + kin * ( pleaf -
p[stomY[k]][stomX[k]] ) );
            if( bStom ) {           //If auto is checked for stomata, want min and max J
                minJ = min( minJ, Jout[k] );
                maxJ = max( maxJ, Jout[k] ); }
            }
        if( bAuto ) {
            minp = FLT_MAX;      //If auto is checked, want min and max p
            maxp = -FLT_MAX; }
        meanp = 0;
        for( i=0; i<nrows; i++ )
            {
            for( j=0; j<ncols; j++ )
                {
                p[i][j] = pnew[i][j];   //Overwrite old p values with new p values
                meanp += p[i][j];
                if( bAuto ) {
                    minp = min( minp, p[i][j] );
                    maxp = max( maxp, p[i][j] ); }
                }
            }
        meanp /= ncells;
        return;
    }
```

The MoveFull function: This function allows for a more rapid stepping through the automata process. The procedure is to take a large number of steps before updating the display. The number of steps between display updates varies so that the display will appear to update every few seconds. For a 50 x 50 grid of cells, 10,000 steps are taken whereas for a 800 x 800 grid, 1000 steps are taken between display updates.

The MoveFull function is implemented as a new execution thread. A mutex semaphore is used to prevent clashes with access to the data arrays between the move function and the display repaint process.

Display repainting: This occurs when the program receives a WM_PAINT message, indicating that the display should be repainted. All of the grid cells are drawn as filled rectangles using a grayscale value based on the cell value of p and the range of minp to maxp. The scale for the grayscale is the variable fscale = 255.0F / (float)(maxp – minp). A set of 256 grayscale brushes (equal RGB values 0 to 255) was created earlier (as part of the WM_CREATE message response). Here is the program section that draws the grayscale cells:

```
for( i=0; i<nrows; i++ )                    //Draw all cells
   {
      for( j=0; j<ncols; j++ )
         {
            if( (j % 2) == 0 ) cyStart = 8;
            else cyStart = 8 - cySquare / 2;
            rt.top = cyStart + i * cySquare;    //rt is map cell location
            rt.bottom = rt.top + cySquare;
            rt.left = cxStart + j * cxSquare;
            rt.right = rt.left + cxSquare;
            if( p[i][j] <= minp ) iColor = 0;          //make black
            else if ( p[i][j] >= maxp ) iColor = 255; //make white
            else iColor = (int)( (p[i][j] - minp) * fScale );     //scale color
            FillRect( hdc, &rt, hBrush[iColor] );
         }
   }
```

While the grayscale cells in the model have various pixel sizes depending on the dimensions of the grid, the stomata are always drawn the same size regardless of the grid dimensions (4 x 4 pixels). Stomata colored rectangles are drawn after (on top of) the grayscale cells and so if the grid is either 50 x 50 or 100 x 100, the gray outline around the stomata reflects the p value of that stomata. For the larger grid sizes, the stomata rectangle is larger than the pixel dimensions of the grayscale cells.

The map of the grid is fixed at 800 x 800 pixels, to make cell locations easier to determine. Therefore for a 800 x 800 cell grid, each cell is one pixel in size. For a 50 x 50 cell grid, each cell is 16 x 16 pixels.

The grid and mapping to the array of p values:

Earlier automata versions considered cells with 4 or 8 neighbors. In either case, the grid appearance mapped directly to a 2D array of values for p with rows and columns matching the grid appearance. The idea arose that a cell might have not 4 or 8 connected neighbors but 6 (thanks to David Costa). I think he was thinking of hexagons as the largest sided regular polygon that was space-filling (tessellating, tiling).

For cells as hexagons, the grid to array mapping is not so simple. In that case, each cell is in contact with and exchanges flow with 6 adjoining cells. The physical map (Figure 1) shows the hexagonal cells and adjacent contacting cells. The cells values of P form a 2D square array. In order to map between the physical and numerical array representations, the cell columns are divided into even and odd rows numbers.

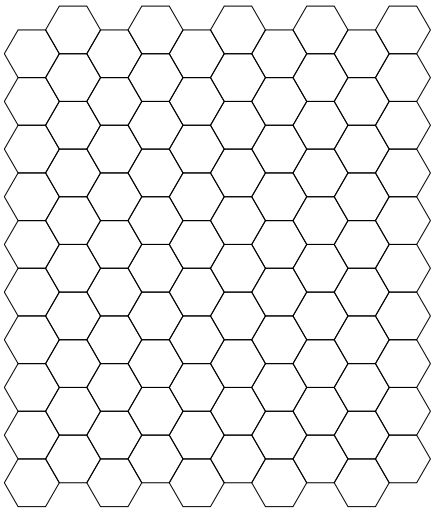


Figure 1. Physical map of a section of the cells.

The cell values are $P_{i,j}$ where i is the row number and j is the column number. An example of this arrangement has the columns forming straight lines but the rows have offset cells horizontally across the grid (Figure 2).

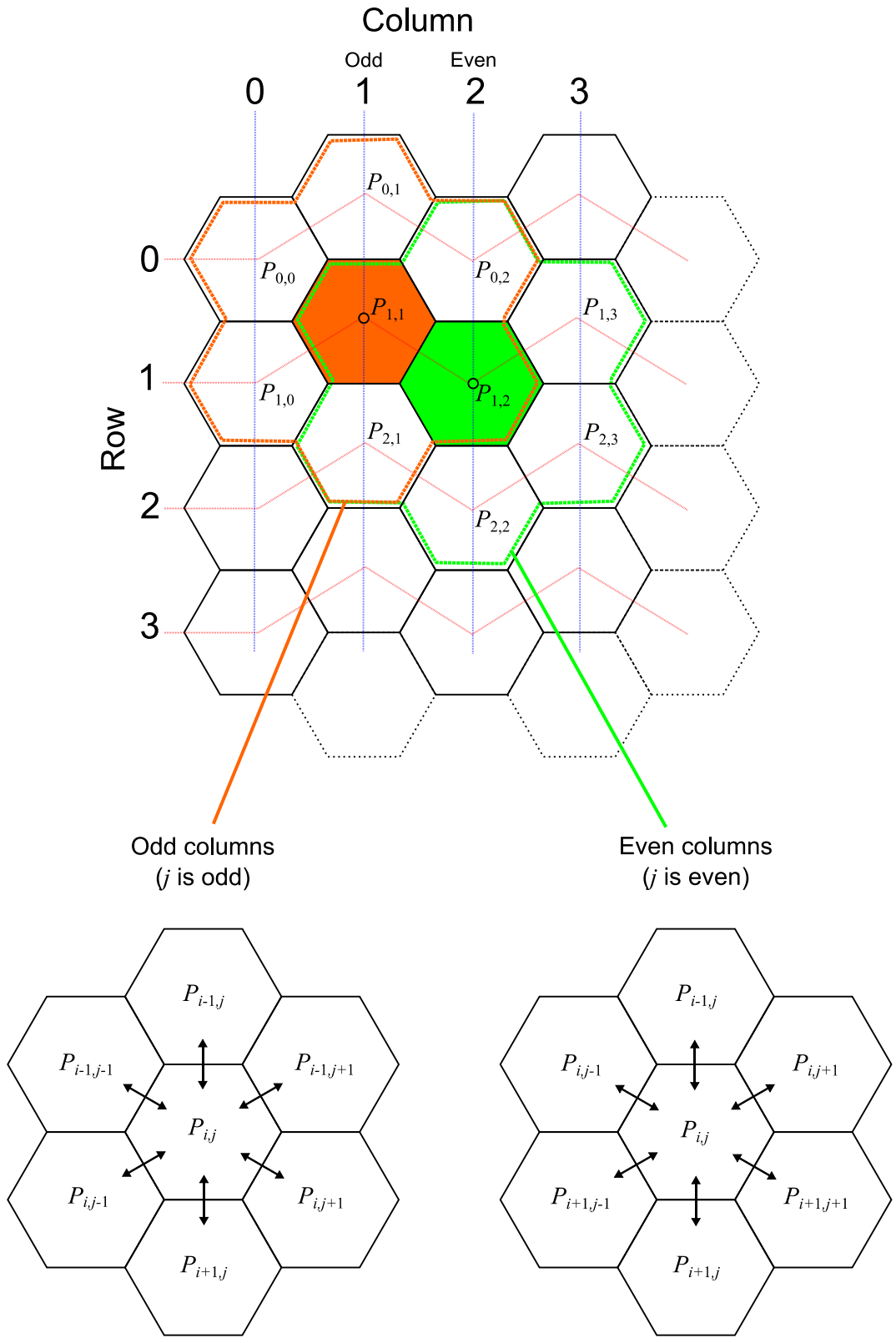


Figure 2. Mapping between the physical arrangement of hexagons and the numerical data array ($P_{i,j}$ where i is the row and j is the column in the array). The upper part

shows the upper left corner region of the grid. Row numbers are shown on the left side along with the red dotted lines connecting cells in the same row. Column numbers are shown along the top along with blue dotted lines connecting cells in the same column. The cell highlighted in orange labeled $P_{1,1}$ is in an odd numbered column. Flow can occur between that cell and its six neighbors (highlighted in orange). The numerical array members for these flows are shown in the bottom left figure for cells in odd columns. The cell highlighted in green labeled $P_{1,2}$ is in an even numbered column. Flow can occur between that cell and its six neighbors (highlighted in green). The numerical array members for these flows are shown in the bottom right figure for cells in even columns.

References

Schulte, P.J., Costa, D.G. 1996. A mathematical description of water flow through plant tissues. Journal of Theoretical Biology 180, 61-70.

Smith, J.A.C., Nobel, P.S. 1986. Water movement and storage in a desert succulent: Anatomy and rehydration kinetics for leaves of *Agave deserti*. Journal of Experimental Botany 37, 1044-1053.