BGHiggins/UCDavis/Revised May 2013

Finite Difference Analysis of Reaction Diffusion Systems: Turing Patterns

Background

The analysis of Turing patterns in the x-y plane requires solving parabolic equations in two spatial dimensions. The typical structure of these equations is

$$\frac{\partial u}{\partial t} = K_1 \nabla^2 u + f(u, v), \quad 0 < x < L_x, \; 0 < y < L_y, \; t > 0$$
$$\frac{\partial v}{\partial t} = K_2 \nabla^2 u + g(u, v), \quad 0 < x < L_x, \; 0 < y < L_y, \; t > 0 \tag{1}$$

The boundary conditions may be Dirichlet, Neumann, Robin or Periodic conditions. In these notes we will consider as finite difference method known as a FTCS method-forward in time, central in space. This is an explicit method. By that we mean the variables at t+1 step can be determined from the t time step without solving a system of equations. The one draw back of this method is that stability of the numerical method is not assured. In general, one has to take a sufficient small step size to ensure that round-off errors do not propagate in time and thereby contaminate the numerical solution.

But as we will show the programming requirements are not very onerous, and involve few lines of code, if one works with lists and use the *Mathematica* functions RotateLeft and RotateRight.

1-D Heat Conduction

We consider the transient heat conduction in a rod of length L. We assume that the conduction of heat along the rod is 1- dimensional and that the rod is insulated except for its ends and there are no heat sources. The partial differential equation (PDE) that describes the transport of thermal energy is given by

$$\frac{\partial u}{\partial t} = K \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < 1, \; t > 0 \tag{2}$$

subject to the following initial and boundary conditions

$$\begin{aligned} &\text{IC}: && u(x, 0) = f(x) = \text{Sin}(x) \\ &\text{BC1}: && u(0, t) = 1 \\ &\text{BC2}: && u(1, t) = 2 \end{aligned} \tag{3}$$

In this problem we have assumed that the diffusivity in Eq. (2) does not depends on the spatial location along the rod. Although it is possible to solve (1)-(2) using the method of separation of variables, we will demonstrate how a numerical technique based on an explicit finite difference method (FTCS) can be used to solve these equations.

Suppose we are given a function $u(x)$ defined on the domain $a \le x \le b$. We can subdivide the domain into a grid of N intervals such that the grid coordinates are $x_1 \equiv a + i\,\triangle x$, where $i = 1, 2, \ldots, N$ and $\triangle x = (b - a)/N$. With this definition we then have $x_0 = a$, and $x_N = b$. The function evaluated at the grid coordinates is written as

$$u_{i+1} = u(x_i + \triangle x),\ u_{i-1} = u(x_i - \triangle x),\ u_i = u(x_i)\ \text{etc} \tag{4}$$

Then the central difference approximation for the second derivative at node x_i becomes

$$\frac{d^2 u}{dx^2}(x_i) = \frac{u_{i+1} - 2\,u_i + u_{i-1}}{\triangle x^2} \tag{5}$$

We can readily extend these definitions to functions of more than one variable. We will show this development in a later section.

For the transient heat conduction in a rod the temperature u depends on both x and t: thus we write $u(x, t)$. We subdivide the rod into a grid of $N - 1$ intervals such that the grid coordinates are $x_i = (i - 1)\,\triangle x$, where $i = 1, 2, \ldots N$, and $\triangle x = L/(N - 1)$. With this definition it follows that $x_1 = 0$, and $x_N = L$. Next we discretize time such that at the n-th time step, the variable time is approximated as $t_n = n\,\triangle t$, where $\triangle t$ is the time interval. Then the temperature evaluated at a given grid point at a given time step is given by $u(x_i, t_n)$. We will use the following nomenclature to represent the *nodal* values of the temperature at the grid points along the rod

$$u_i^n = u(x_i, t_n),\ u_i^{n+1} = u(x_i, t_{n+1})\ \text{etc} \tag{6}$$

If we take a forward difference of the time derivative the finite difference approximation on the node x_i at t_n becomes

$$\frac{u_i^{n+1} - u_i^n}{\triangle t} = K\,\frac{(u_{i+1}^n - 2\,u_i^n + u_{i-1}^n)}{\triangle x^2} \tag{7}$$

Rearranging terms, we obtain the following explicit equation for the temperature at the n+1 time step in terms of the temperature at the n-th time step

$$u_i^{n+1} = u_i^n + \frac{K\,\triangle t}{\triangle x^2}\left(u_{i+1}^n - 2\,u_i^n + u_{i-1}^n\right) \tag{8}$$

Recall from the boundary conditions (3) that following conditions must be satisfied at the n-th time step:

$$u_i^n = 1,\ u_N^n = 2 \tag{9}$$

This gives the following set of equations for the temperature at the $n + 1$ time step

$$\begin{aligned} u_1^{n+1} &= 1 \\ u_i^{n+1} &= u_i^n + \frac{K\,\triangle t}{\triangle x^2}\left(u_{i+1}^n - 2\,u_i^n + u_{i-1}^n\right),\ i = 2, \ldots, N - 1 \\ u_N^{n+1} &= 2 \end{aligned} \tag{10}$$

The system of equations given by (9) is an explicit formula for determining the unknown nodal values at the n+1 time step *if we know the nodal values at the* n^{th} *time step*. Note that the BCs are used to ensure that the nodal values at $i = 1$ and $i = N$ are satisfied for all time $n > 1$. To start the calculation we invoke the initial condition at $t = 0$

$$u_i^1 = f(x_i),\ i = 1, 2, \ldots, N \tag{11}$$

Note that we may have a discontinuity in the nodal values at i=1 and i=N as

$$u_1^1 \neq u_1^n,\ u_N^1 \neq u_N^n,\ \text{for } n > 1 \tag{12}$$

Normally this discontinuity gets smoothed out rapidly if Δt is sufficiently small. The above numerical method is not unconditionally stable. One can show for numerical stability that

$$\frac{\triangle t\, K}{\triangle x^2} \leq \frac{1}{2} \tag{13}$$

Thus as we make the grid mesh finer we have to reduce the time step so that Eq. (13) is not violated. For this reason explicit FTCS methods are not very efficient for solving complicated problems.

Mathematica Programming

There are many ways to program the explicit method. Here we will illustrate a technique based on the functions **RotateLeft and RotateRight** for manipulating lists that allow the computations to be done "in parallel" by operating on the full list at every time step. The method is adapted from techniques described by Glass(1998) and by Shaw and Ting(1994). As we shall see later in this notebook, these techniques can be readily extended to higher dimensional problems, namely the reaction diffusion equation with periodic boundary conditions.

Let us specify the various grid parameters

```
Δx = 1 / 20; K = 0.5; Δt = 0.48 Δx^2 / K;
```

Note we have selected the time step so that $\Delta t\, K / \Delta x^2 < 0.5$ to ensure numerical stability. Next we define the initial condition. We do this by simply mapping the function f(x) over the grid

```
initCond = Map[Sin[π #] &, Range[0, 1, Δx]];
```

Thus the length of our list is

```
Length[initCond]
```

```
21
```

The last step is to define the function that creates the finite difference template. Here we use RotateLeft and RoteRight (one can also use ListConvolve) to generate the template for the spatial derivative. To see how this works consider a test array

```
u = Array[U, 6]
```

```
{U[1], U[2], U[3], U[4], U[5], U[6]}
```

Then if we apply RotateLeft and Rotate Right functions to this array we get

```
b1 = (RotateRight[ u] + RotateLeft[u] - 2 u) / h^2
```

$$\left\{\frac{-2\,U[1]+U[2]+U[6]}{h^2}, \frac{U[1]-2\,U[2]+U[3]}{h^2}, \frac{U[2]-2\,U[3]+U[4]}{h^2}, \frac{U[3]-2\,U[4]+U[5]}{h^2}, \frac{U[4]-2\,U[5]+U[6]}{h^2}, \frac{U[1]+U[5]-2\,U[6]}{h^2}\right\}$$

Each element of the above list is the central difference approximation of the spatial derivative in our PDE with a grid spacing of h applied at the grid points $i = 1, , 2, \ldots, 6$. The first and last elements (i=1, i=6) are incorrect since the BCs at those nodes must be specified-see Eq. (10). We can fix that during the iteration. Let us define a function that evaluates Eq. (10) at each iteration.

```
DiffusionIter[K_, Δt_, Δx_, u_] := Module[{a, b, c},
  b = u + (Δt K)/Δx^2 (RotateRight[u] + RotateLeft[u] - 2 u);
  b[[1]] = 0; b[[Length[b]]] = 1; b]
```

Note how after the elements are computed the first and last elements are set equal to the values defined by the BCs. It is important to note here that the variables b, u appearing in the above function are lists. Thus we are doing list operations: by adding two lists we add the individual elements of those lists.

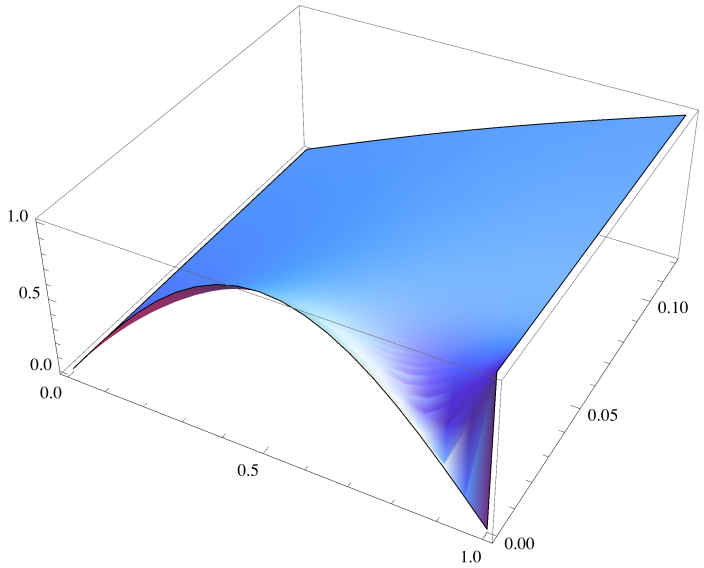
To perform the necessary iterations in time, we make use of **NestList** This generates a nested list of u[i] values at each time step Here is the list of values for 200 iterations

```
res = NestList[DiffusionIter[K, Δt, Δx, #] &, initCond, 200];
```

Note how we start out our iteration using the list of initial conditions.

To plot the values we use ListPlot3D. This generates the solution in both time and space

```
ListPlot3D[res, Mesh → False, DataRange → {{0, 1}, {0, Δt 50}}]
```



For the above coding we have no For loops. The algorithm is based on manipulating the elements of the list- in this case the nodal values of the temperature all at once.

Reaction Diffusion Equation

Suppose we want to solve the following reaction diffusion equation with periodic boundary condition by an explicit FD method

$$\frac{\partial u}{\partial t} = K \nabla^2 u + f(u), \quad 0 < x, y < L, \quad t > 0 \tag{14}$$

The initial condition is

$$\text{IC}: \quad u(0, x, y) = f(x, y) \tag{15}$$

while the periodic boundary conditions are

PBC1 $u(t, x, 0) = u(t, x, L)$

PBC2 : $\frac{\partial u}{\partial y}(t, x, 0) = \frac{\partial u}{\partial y}(t, x, L)$

PBC3 $u(t, 0, y) = u(t, L, y)$

PBC2 : $\frac{\partial u}{\partial x}(t, 0, y) = \frac{\partial u}{\partial y}(t, L, y)$

In this case our spatial grid is 2-D and for convenience we take the grid spacing to be uniform in both x and y. Thus our finite difference approximation of the PDE at the x_i, y_j node at time $t_{n+1}$is

$$u_{i,j}^{n+1} = u_{i,j}^n + \Delta t\, f\left(u_{i,j}^n\right) + \frac{K\,\Delta t}{\Delta x^2}\left(u_{i+1,j}^n - 2\,u_{i,j}^n + u_{i-1,j}^n\right) + \frac{K\,\Delta t}{\Delta y^2}\left(u_{i,j+1}^n - 2\,u_{i,j}^n + u_{i,j-1}^n\right) \tag{17}$$

Finite Difference Template

To see how we generate the finite difference template consider a grid that is 4×4 given by

```
u = Array[U, {4, 4}]; u // MatrixForm
```

$$\begin{pmatrix} U[1,1] & U[1,2] & U[1,3] & U[1,4] \\ U[2,1] & U[2,2] & U[2,3] & U[2,4] \\ U[3,1] & U[3,2] & U[3,3] & U[3,4] \\ U[4,1] & U[4,2] & U[4,3] & U[4,4] \end{pmatrix}$$

Note using our notation above the first index corresponds to x and the second to y. Thus rows represent x-values , columns represent y values.

We can apply the RotateRight and RotateLeft functions to each row of the matrix to obtain the spatial discretization of u in the y- direction. We do this by specifying the second argument of RotateLeft and RotateRight

```
b1 = (RotateRight[u, {1, 0}] + RotateLeft[u, {1, 0}] - 2 u) / h^2;
```

Let us look at the first row of that matrix

```
b1[[1]]
```

$$\left\{\frac{-2\,U[1,1] + U[2,1] + U[4,1]}{h^2}, \frac{-2\,U[1,2] + U[2,2] + U[4,2]}{h^2}, \frac{-2\,U[1,3] + U[2,3] + U[4,3]}{h^2}, \frac{-2\,U[1,4] + U[2,4] + U[4,4]}{h^2}\right\}$$

Thus each row of the above matrix is the discretization of $\left(\partial^2 u/\partial x^2\right)_{i,j}$ using central differencing. Note the first and last element involve wrapping the list about itself. This is how we satisfy periodic BCs.

Likewise to get the spatial discretization in the y-direction we use

```
b2 = (RotateRight[u, {0, 1}] + RotateLeft[u, {0, 1}] - 2 u) / h^2;
```

```
b2[[1]]
```

$$\left\{\frac{-2\,U[1,1]+U[1,2]+U[1,4]}{h^2},\ \frac{U[1,1]-2\,U[1,2]+U[1,3]}{h^2},\ \frac{U[1,2]-2\,U[1,3]+U[1,4]}{h^2},\ \frac{U[1,1]+U[1,3]-2\,U[1,4]}{h^2}\right\}$$

Now if we add up the first two rows

```
b1[[1]] + b2[[1]] // Simplify
```

$$\left\{\frac{-4\,U[1,1]+U[1,2]+U[1,4]+U[2,1]+U[4,1]}{h^2},\ \frac{U[1,1]-4\,U[1,2]+U[1,3]+U[2,2]+U[4,2]}{h^2},\ \frac{U[1,2]-4\,U[1,3]+U[1,4]+U[2,3]+U[4,3]}{h^2},\ \frac{U[1,1]+U[1,3]-4\,U[1,4]+U[2,4]+U[4,4]}{h^2}\right\}$$

each element is the spatial discretization of $\nabla^2 u$ along the first row of the grid: $\{i = 1,\ j = 1, 2, \ldots, 4\}$

Consider the node i=2,j=2. Then the Laplacian becomes

$$\left(\nabla^2 u\right)_{2,2} = \frac{(u_{1,2}+u_{3,2}+u_{2,1}+u_{2,3}-4\,u_{2,2})}{\triangle x^2} \tag{18}$$

The above expression is readily found from the elements at position i=2,j=2 in b1 and b2 as shown below

```
(b2[[2, 2]] + b1[[2, 2]]) // Simplify
```

$$\frac{U[1,2]+U[2,1]-4\,U[2,2]+U[2,3]+U[3,2]}{h^2}$$

Thus we can construct the following ReactionDiffusion iterator

```
ReactionDiffusionIter[Dx_, Dy_, Δt_, Δx_, f_, u_] := Module[{},
  Flatten[u + Δt Dx/Δx^2 (RotateRight[u, {1, 0}] + RotateLeft[u, {1, 0}] - 2 u) +
    Δt Dy/Δy^2 (RotateRight[u, {0, 1}] + RotateLeft[u, {0, 1}] - 2 u) + Δt f[u]]]
```

As we will see below the periodic boundary conditions are automatically satisfied. We use Flatten so that our N×N grid becomes a 1-D list of length N^2

Periodic Boundary Conditions

Now consider the element how the periodic boundary conditions are handled. Consider the nodal values of u along the x grid at say y=j

$$\{u_{1,j},\ u_{2,j},\ u_{3,j},\ \ldots,\ u_{5,j},\ u_{6,j}\} \tag{19}$$

Thus if we evaluate the spatial derivative $\partial^2 u / \partial x^2$ at say i=1, we get

$$\left(\frac{\partial u}{\partial x}\right)_{1,j} = \frac{u_{0,j} - 2\,u_{1,j} + u_2}{\triangle x^2} \tag{20}$$

where now $u_{0,j}$ is a fictitious node to the left of $u_{1,j}$. But with periodic boundary conditions this must be equal to $u_{6,j}$ as we have the following repeating pattern if the system is to be periodic

$$\begin{aligned}&\{u_{1,j},\ u_{2,j},\ u_{3,j},\ \ldots,\ u_{5,j},\ u_{6,j},\ u_{1,j},\\ &u_{2,j},\ u_{3,j},\ \ldots,\ u_{5,j},\ u_{6,j},\ u_{1,j},\ u_{2,j},\ u_{3,j},\ \ldots,\ u_{5,j},\ u_{6,j}\}\end{aligned} \tag{21}$$

Thus the list wraps onto itself. Thus

$$\begin{aligned}\left(\frac{\partial u}{\partial x}\right)_{1,j} &= \frac{u_{6,j} - 2\,u_{1,j} + u_{2,j}}{\triangle x^2}\\ \left(\frac{\partial u}{\partial x}\right)_{6,j} &= \frac{u_{1,j} - 2\,u_{6,j} + u_{5,j}}{\triangle x^2}\end{aligned} \tag{22}$$

Hence our method for creating the finite difference template automatically generates the appropriate periodic boundary conditions!

Test Code

We will consider a 4×4 grid to inspect how our reaction diffusion iterator behaves

```
Remove[Δt, Δx];
u = Array[U, {4, 4}]
```

```
{{U[1, 1], U[1, 2], U[1, 3], U[1, 4]}, {U[2, 1], U[2, 2], U[2, 3], U[2, 4]},
 {U[3, 1], U[3, 2], U[3, 3], U[3, 4]}, {U[4, 1], U[4, 2], U[4, 3], U[4, 4]}}
```

Let us suppose that the function f is given by

```
f[v_] := v (1 - v)
```

Now we can evaluate the

```
res = ReactionDiffusionIter[Dx, Dy, Δt, Δx, f, u];
```

As expected the number of finite difference equations for this grid should be 16

```
Length[res]
```

```
16
```

Here is what the first two equations at i={1,2} j=1 look like

```
res[[{1, 2}]]
```

$$\begin{aligned}\Big\{&U[1,1] + \triangle t\,(1 - U[1,1])\,U[1,1] + 2500\,Dy\,\triangle t\,(-2\,U[1,1] + U[1,2] + U[1,4]) +\\ &\frac{Dx\,\triangle t\,(-2\,U[1,1] + U[2,1] + U[4,1])}{\triangle x^2},\ U[1,2] + \triangle t\,(1 - U[1,2])\,U[1,2] +\\ &2500\,Dy\,\triangle t\,(U[1,1] - 2\,U[1,2] + U[1,3]) + \frac{Dx\,\triangle t\,(-2\,U[1,2] + U[2,2] + U[4,2])}{\triangle x^2}\Big\}\end{aligned}$$

Let us do a simple test with the following parameters

```
Δx = Δy = 1 / 20; Dx = Dy = 0.5; Δt = 0.48 Δx^2 / Dx;

IC = Table[0.2, {20}, {20}];

res =
  NestList[Partition[ReactionDiffusionIter[Dx, Dy, Δt, Δx, f, #], 20] &, IC, 1000];
```

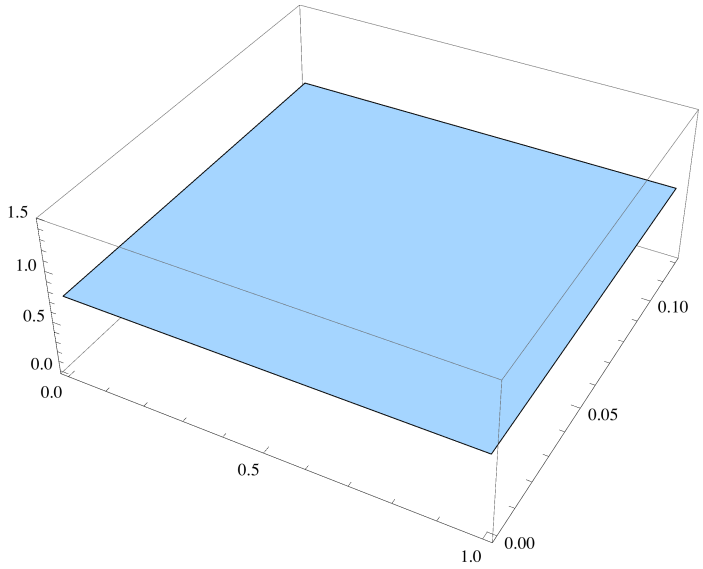
The solution for each time step is stored in the variable res. Note that the output from each iteration of Nest lest is partitioned and then fed back as the new "initial condition". Here is the dimension of the array

```
Dimensions[res]
```

```
{1001, 20, 20}
```

Here is a plot at the 1000th time step

```
ListPlot3D[res[[1000]], Mesh → False, DataRange → {{0, 1}, {0, Δt 50}}]
```



In this case a constant solution at each time step satisfies the PDE

Extension to Multiple PDES

Suppose now we have the following system of PDEs

$$\frac{\partial u}{\partial t} = F(u, v) + \mathcal{D}_a \nabla^2 u$$

$$\frac{\partial v}{\partial t} = G(u, v) + \mathcal{D}_b \nabla^2 v \tag{23}$$

The functions F and G are given by

$$F = \frac{u^2}{v} - u, \quad G = u^2 - v \tag{24}$$

For each species we have the following operator

```
ReactionDiffusionIter1[Du_, Δt_, Δx_, f_, u_, v_] := Module[{},
  Flatten[u + (Δt Du)/Δx^2 (RotateRight[u, {1, 0}] + RotateLeft[u, {1, 0}] - 2 u) +
    (Δt Du)/Δy^2 (RotateRight[u, {0, 1}] + RotateLeft[u, {0, 1}] - 2 u) + Δt f[u, v]]]

ReactionDiffusionIter2[Dv_, Δt_, Δx_, g_, u_, v_] := Module[{},
  Flatten[v + (Δt Dv)/Δx^2 (RotateRight[v, {1, 0}] + RotateLeft[v, {1, 0}] - 2 v) +
    (Δt Dv)/Δy^2 (RotateRight[v, {0, 1}] + RotateLeft[v, {0, 1}] - 2 v) + Δt g[u, v]]]
```

We take the following parameters and initial conditions for this simulation

```
Δx = Δy = 1 / 50;
Du = .0005; Dv = .01; Δt = .20 * Δx^2 / Max[{Du, Dv}];
ICu = Table[Random[], {y, 0, 1, Δx}, {x, 0, 3 / 2, Δx}];
ICv = Table[.1 + Random[], {y, 0, 1, Δx}, {x, 0, 3 / 2, Δx}];
```

Let us check the dimensions of the grid

```
Dimensions[ICu]
```

```
{51, 76}
```

We define the following functions

```
F[u_, v_] := u^2/v - u
G[u_, v_] := u^2 - v
```

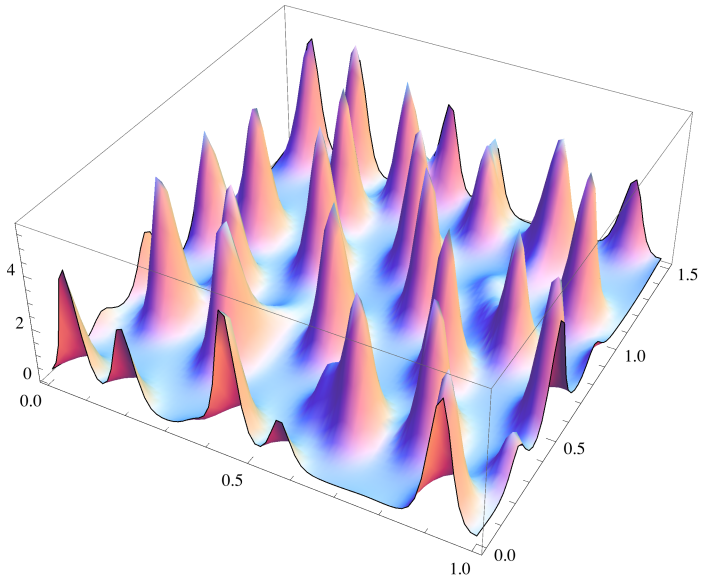
For this problem we will select these parameters and take a grid of 51×76

To iterate this system we use NestList. Note the initial condition is now a list of two initial conditions (one each for u and v) . Each element of this list is a sublist with dimensions 51×76. These are fed to each iteration function Here is calculation with 1000 time steps- it takes 120 seconds on G5 Mac (1.6 MHz)

```
res = NestList[{Partition[ReactionDiffusionIter1[Du, Δt, Δx, F, #[[1]], #[[2]]], 76],
      Partition[ReactionDiffusionIter2[Dv, Δt, Δx, G, #[[1]], #[[2]]],
       76]} &, {ICu, ICv}, 1000];
```

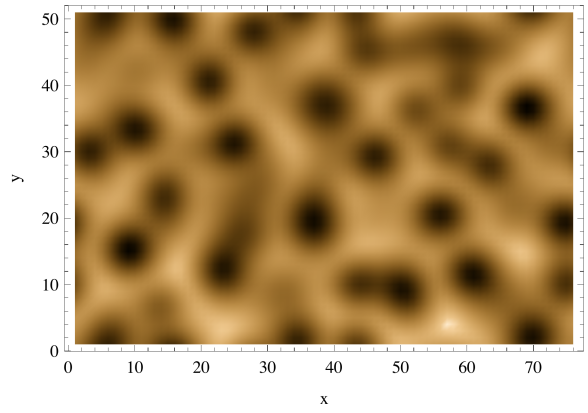
Let us plot the result for u at the 1000th time step

```
ListPlot3D[res[[1000, 1]], Mesh → False,
 DataRange → {{0, 1}, {0, 3 / 2}}, PlotRange → All]
```



Here is a density plot of the patterns that illustrates “leopard spots”: I have made an effort to pick a color function that illustrates “spots”. The function is quite sensitive to the sharpness of the peaks visible in the previous plot.

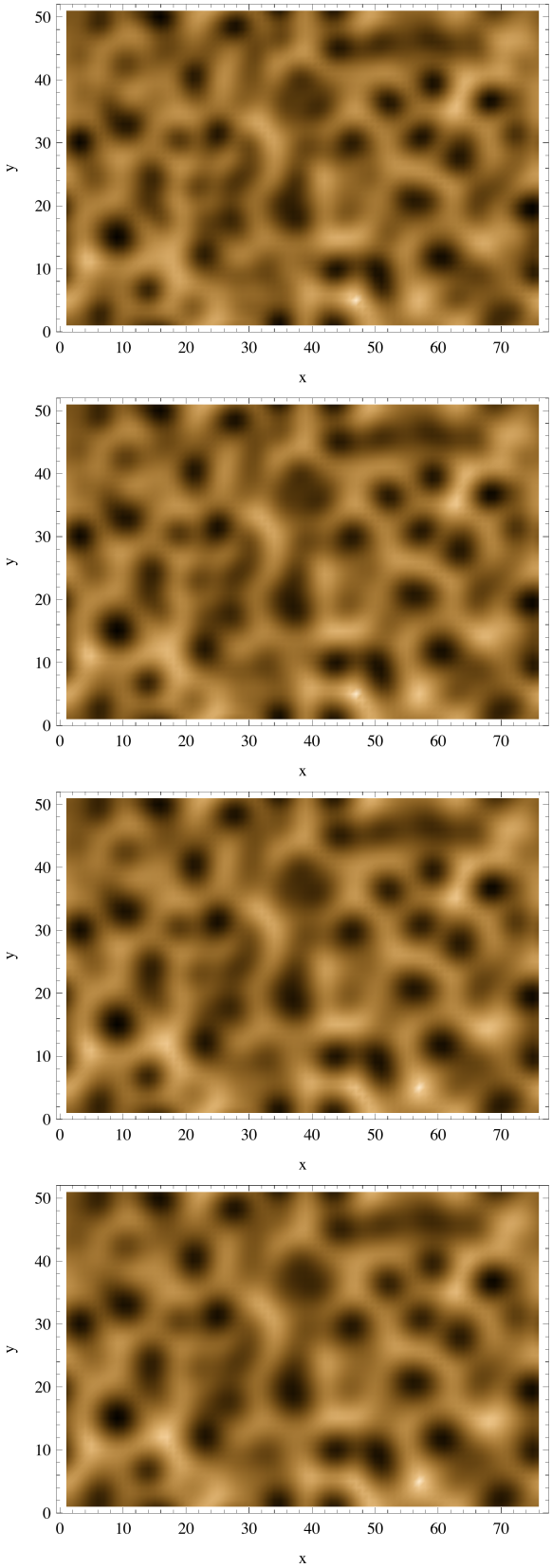
```
ListDensityPlot[res[[700, 1]], ColorFunction → (Hue[.1, .2 + .8 Sqrt[Sqrt[#]], 1 - .98 Sqrt[#]] &),
 Frame → True, FrameLabel → {"x", "y"}, ColorFunctionScaling → True,
 ImageSize → 300, Mesh → False, AspectRatio → Automatic]
```

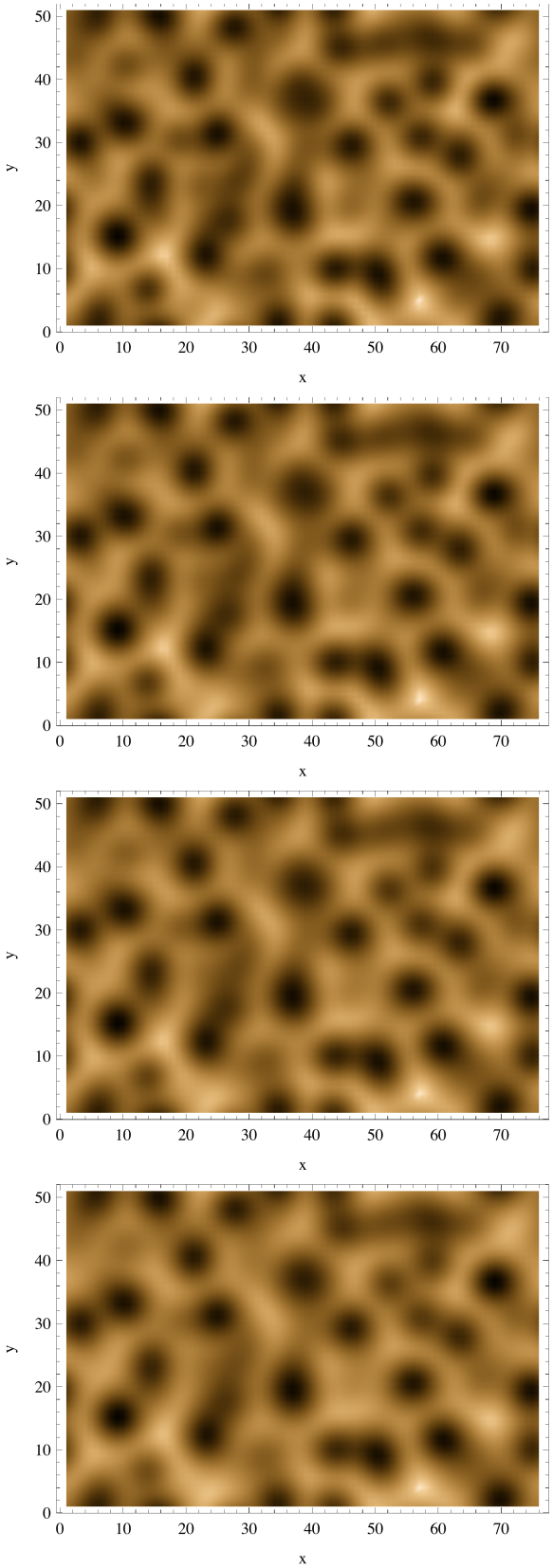


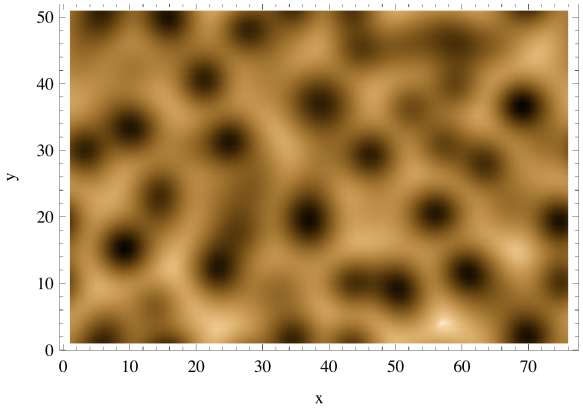
Let us show an animation of the evolution of the pattern as a function of time in steps of 50 time steps

```
myplots[n_] := ListDensityPlot[res[[n, 1]],
   ColorFunction → (Hue[.1, .2 + .8 Sqrt[Sqrt[#]], 1 - .98 Sqrt[#]] &), Frame → True,
   FrameLabel → {"x", "y"}, ColorFunctionScaling → True, ImageSize → 300,
   Mesh → False, AspectRatio → Automatic, MaxPlotPoints → 100, Mesh → All];
```

```
Column[Map[myplots[#] &, Range[300, 700, 50]]]
```




y
x
0
10
20
30
40
50
60
70



Comments

I have not explored in detail the accuracy of the code. What is important is that the stability criterion for the 2-D system, which is more restrictive than the 1-D case must be maintained. That is

$$\frac{\triangle t\, D}{\triangle x^2} < 0.25$$

An alternative method is to used NDSolve to solve the finite difference equations using a method of lines approach, which removes the stability criterion of a simple explicit time stepping approached used in this study.

References

The code development in this notebook is based on ideas developed in the following references.

R. Glass, *Mathematica for Scientists and Engineers*, Prentice Hall,1998

W. T. Shaw & J. Tigg, Applied *Mathematica*. Getting Started, Getting Done. Addison-Wesley,1994