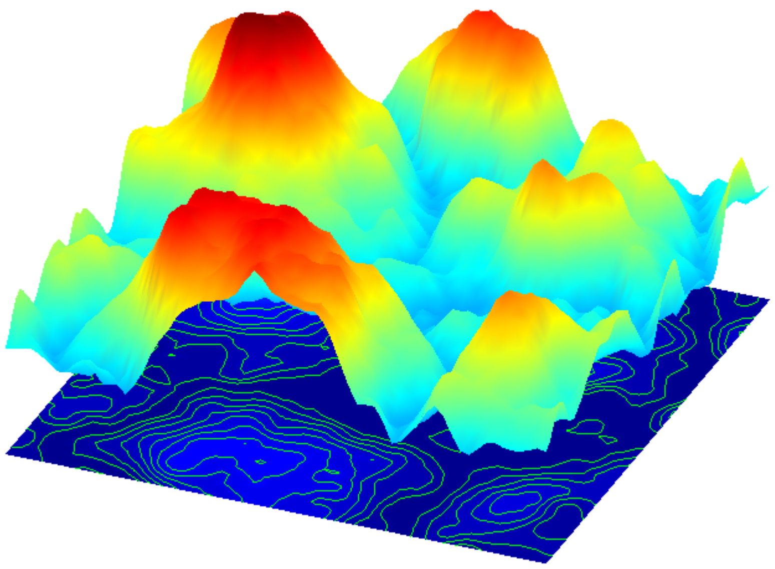
Spatiotemporal Models in Ecology: An Introduction to Integro-Difference Equations

James Powell[*]
Department of Mathematics and Statistics
Utah State University

April 21, 2009



[*]jim.powell@usu.edu

Contents

Additional Material Included in Reader

- *Portions of Getting Started with* MATLAB. (yellow)
- Allen, J.C., C.C. Brewster and D.H. Slone, 2001. "Spatially explicit ecological models: a spatial convolution approach," *Chaos, Solitons and Fractals* 12: 333–347. (white)
- Andersen, M. 1991. "Properties of some density-dependent integrodifference equation population models," *Mathematical Biosciences* 104: 135-157. (pink)
- Edelstein-Keshet, L. 1998. *Mathematical Models in Biology* Random House/Birkhäuser Mathematics Series, NY, pp. 78-89. (white)
- Holmes, E.E., M.A. Lewis, J.E. Banks and R.R. Veit, 1994. "Partial differential equations in ecology: spatial interactions and population dynamics," *Ecology* 75: 17-29 (from the special issue *Space, the Final Frontier for Theoretical Ecology*). (pink)
- Kot, M.A. Lewis and P. v.d. Driessche, 1996. Dispersal data and the spread of invading organisms. *Ecology* 77: 2027–2042. (white)
- Neubert, M.G., M. Kot and M.A. Lewis, 1995. "Dispersal and pattern formation in a discrete time predator-prey model," *Theoretical Population Biology* 48: 7–43. (pink)
- Powell, J.A. and N.E. Zimmermann, 2004. "Multi-Scale Analysis of Active Seed Dispersal Contributes to Resolving Reid's Paradox," *Ecology* 85:490-506. (white)
- Powell, J.A., T. McMillen and P. White, 1998. "Connecting a Chemotactic Model for Mass Attack to a Rapid Integro-Difference Emulation Strategy," *SIAM Journal of Applied Mathematics* 59: 547-572. (pink)

Approximate Class Syllabus 2009

First Day, Wednesday, 6 May

09:00 Class introduction. Spatial modelling survey and introduction to mathematical modelling concepts involving space.

10:15 Coffee!

10:30 Qualitative behavior and derivation of some important spatial and temporal models.

11:45 Lunch!

13:00 Computer Lab 1: Introduction to MATLAB including operations, variables, matrices, plotting and graphics, help and documentation.

14:30 Coffee!

16:30 End of computer lab.

Second Day, Thursday, 7 May

09:00 Discussion of random walks and the diffusion equation. Gaussian dispersal, use of Fourier Transforms to solve the diffusion equation, introduction to Discrete and Fast Fourier Transforms as numerical tools.

10:15 Coffee!

10:30 Examples of dispersal kernels: drift-diffusion (advection-diffusion), diffusion and settling, 'get out of here' diffusion, ballistic dispersal, prey-taxis and differential motility.

11:45 Lunch!

13:00 Computer Lab 2: Using MATLAB and FFT to simulate dispersal in one and two dimensions.

14:30 Coffee!

16:30 End of computer lab.

Third Day, Monday, 11 May

09:00 Disscussion on implementing integro-difference models: presentation and discussion of several literature case-studies. Spread of plants, Allee effects, host-parasite dynamics in space, seed shadows, splash-dispersal of fungi, fungal pandemics, outbreaks of tree-killing beetles, *Drosophila* in apple orchards

10:15 Coffee!

10:30 Groups organize case studies. Define phenomena to model, questions to answer, formulate models for discrete and dispersal dynamics, discuss parameterization, divide work responsibilities.

11:45 Lunch!

13:00 Computer Lab 3: Implementing IDE in MATLAB and simulating prey-taxis.

14:30 Coffee!

16:30 Group work on case-studies if time permits. End of computer lab.

Fourth Day, Wednesday, 13 May

09:00 More case studies of IDE. Discussion of theory of how to implement boundary conditions (reflecting (solid), absorbing (lethal)) using dispersal technology.

10:15 Coffee!

10:30 Modelling chemotaxis, implementation of chemotactic model using IDE approaches. Brief discussion with case study groups.

11:45 Lunch!

13:00 Computer Lab 4: Reflecting and absorbing boundaries in one and two dimensions, the Powell approach to chemotaxis.

14:30 Coffee!

16:30 Group work on case-studies if time permits. End of computer lab.

Fifth Day, Thursday, 14 May

09:00 Answer last questions, finish whatever discussion has not been finished. **Presentation of Case Studies** by groups.

10:15 Coffee!

10:30 **More presentation of Case Studies**

11:45 Lunch and **Even more presentation of Case Studies**

Whenever End of class. Beer for the teacher!

1 Introducing MATLAB

1.1 Goals

The goals of this lab are to introduce students to important MATLAB commands and philosophies, including:

- Basic graphical operations for plotting data in one and two dimensions.
- Defining grids.
- MATLAB's Help facility.
- Special mathematical and logical functions.
- Matrix manipulations (Demos).
- Implementing the Fast Fourier Transform (FFT) in MATLAB (Demos).

MATLAB is built around matrix and vector manipulation, and views everything as vectors or matrices or tensors. It is not a symbolic program, and performs no analytic calculations, but is composed of computationally optimized linear algebra routines for performing operations on and to matrices and vectors.

1.2 Introduction to the Software

1.2.1 Matrices and Basic Operations

The MATLAB program was based originally on a set of computational routines for doing linear algebra, and retains a linear algebra bias in its operations. To define the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

in MATLAB use the command

```
A = [ 1 2 3; 4 5 6; 7 8 9]
```

The *column* vector $\vec{c} = (1, 2, 7)^T$ and *row* vector $\vec{r} = (2, 4, 6)$ can be defined

```
c = [1; 2; 7]
r = [2 4 6]
```

Notice that adding a semicolon (`;`) adds to the number of rows (making the columns longer. To multiply the matrix **A** on the left by the row vector $\vec{r}$ use the MATLAB command `r*A`; to multiply the column vector $\vec{c}$ on the left by **A** use the MATLAB command `A*c`. Now, try these backward and see the error message when MATLAB tries to multiply matrices of incompatible sizes!

MATLAB is a great program for doing matrix operations. For example, to find the eigenvalues of **A** use the command

```
eig(A)
```

Type the `help` command, `help eig`, to see what else the `eig` command is capable of in MATLAB . In particular notice that the command

```
[V,D]=eig(A)
```

returns the eigenvalues of **A** as columns in the matrix **V**. You should have noticed that one of the eigenvalues of **A** is zero; this is because the matrix **A** is singular, that is, the rows are dependent. (In this case, if you take twice the second row and subtract the first, you get exactly the third.) You could determine this using by-hand row-reduction, but for giggles try the following command to get the reduced echelon form of **A**:

```
rref(A)
```

1.2.2 Introducing the Help Facilities

Help documentation is available at the MathWorks Web site

http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml

Open a Web browser and link to the documentation at MathWorks to support this Lab exercise. The 'Getting Started' guide should be there in the left hand column; click on this and then click on 'Matrices and Arrays' in the right hand frame to see some of the supporting examples. If you want to expand your knowledge of MATLAB you may want to work through some of these examples.

Take some time now to work through the Matrix Manipulation section of the Getting Started guide. In particular, familiarize yourself with the following types of operations:

- Refer to the matrix element in the second row and third column using `A(2,3))`.
- Refer to the entire second column using the ':' to denote 'all possible values' as in the command `A(:,2)`. To refer to the entire first row you would use `A(1,:)`.
- Use `sum` to sum along columns of a matrix. How could you sum along rows?

You may also want to check out some of the MATLAB demos. Click on the Help menu and then click on Demos. In the left hand menu of the window that appears click on Matrices and then on Basic matrix operations in the right hand menu. This will generate a slide show which will lead you through several of the basic MATLAB matrix and plotting commands.

Shortcuts *There are a couple of nice shortcuts in* MATLAB *that make things go much quicker. One of them is the* $\uparrow$ *key. This brings back commands that you have typed previously, which can then be edited on the command line. Try using* $\uparrow$ *and see what happens. Also, you can use* $\uparrow$ *to search for the last command that begins with certain letters. So, for example, to see the last command you typed in which begins with* `A =` *type 'A =' on the command line and then hit the* $\uparrow$ *key.*

Another shortcut: Often you do not want to see the output of a command, particularly when it is a long vector or big matrix. To suppress output, follow the MATLAB *command with a semicolon.*

Note: *At any time during a* MATLAB *session you may type* **help COMMAND** *for* MATLAB*'s internal information on how* **COMMAND** *may be used, including examples.* If you do not know what command you are looking for, you may type **lookfor SUBJECT** to get MATLAB to search for commands which refer to your SUBJECT.

EXERCISE 1: Type

```
help linspace
```

and after reading about **linspace** type

```
lookfor space
```

to see what other relevant commands there might be.

1.2.3 Defining One-Dimensional Functions and Graphing

Let's begin by graphing some familiar one-dimensional functions using MATLAB . The first step is to make a vector corresponding to the real axis, say between -5 and 5:

```
x = linspace(-5,5,101);
```

The above command generates a vector with 101 components, named `x`, whose elements are evenly spaced numbers beginning at -5 and ending at 5. The semicolon (;) suppresses output – try the command without it and see what happens! Notice also that we get a spacing of Δx = 1/10 using the above command – the 101 allows us to have the endpoints included in the vector of coordinates.

Now define a function, $f(x) = \sin(x^2)/(1 + x^2)$, which we will then plot. The function f will be a vector, `f`, which contains for every component of the coordinate vector, `x`, a value according to the rule $f(x)$:

```
f = sin( x.^2 )./(1 + x.^2);
```

The notation '.' before an operation means to apply the operation element-by-element to an array; thus

```
x.^2
```

means the element by element square of `x`, or a vector of length 101 with values x^2 for x between -5 and 5 (Look in the *Getting Started with* MATLAB materials for more information on how basic arithmetic operations are implemented). Now, to plot `f` issue the MATLAB command

```
plot(x,f)
```

which will plot the ordered pairs given by `x, f`. Try changing the order of `x` and `f` above to see what happens.

Now, to see how to put multiple plots on one graph and change some of the styles, try the following:

```
plot(x,f,'r*')
hold on, plot(x,1./(1+x.^2),'g'), hold off
xlabel('x'),ylabel('f(x)'),title('Function and Envelope')
```

The 'hold' commands force MATLAB to draw plots without erasing previous plots. One may also plot several functions simultaneously, as in

```
figure(2)
plot(x,cos(x),'r',x,exp(-abs(x)),'b',x,sin(x.^2),'g')
```

Now, just to see what some of the options would be, type

```
help plot
```

which will give you a number of examples, sets of options, and some suggestions for other commands you might find useful. Try some of these just to get the hang of it.

EXERCISE 2: Plot the fundamental solution of the heat equation,

$$u_t = Du_{xx}, \quad u(x,0) = \delta(x)$$

which has the form

$$F(x,t) = \frac{1}{\sqrt{4\pi Dt}} \exp\left[-\frac{x^2}{4Dt}\right],$$

for the following t values: 1, 2, 4. Plot all on the same set of axes, over an interval from -10 to 10, for a D value of .37, and organize the time slices in the colors of the MATLAB spectrum ('r','y','g','b','m'). Label the axes and give the plot a title. The `legend` command allows you to label the different graphs – use `help` and check it out!

1.2.4 Scalar Functions of Two Variables

Defining functions of two variables works with the same philosophy in MATLAB , except that the underlying substrate of independent variables becomes a matrix as opposed to a vector. Let's begin by defining two coordinates, `x` and `y` and then a two-dimensionally extended version of these coordinates:

```
x = .1*[0:100];
y = .05*[-50:50];
[X,Y] = meshgrid( x, y);
```

(Notice the scalar multiplication of a spacing, either .1 or .05 above, and an array of numbers, `[N1:N2]`, which is all of the integers between `N1` and `N2`, inclusive. Thus, `x = .1*[0:100]` is equivalent to `x = linspace(0,10,101).`). Now we may define a function of these two variables analogously to the one-dimensional procedure. For example, to plot the function

$$g(x,y) = \frac{\sin(x^2 + \pi y)}{1 + x^2 + y^2},$$

as a surface, we could use the sequence of commands

```
g = sin( X.^2 + pi*Y )./( 1 + X.^2 + Y.^2 );
surf(X, Y, g)
```

Can you tell which is the x and which the y direction? Now let's change the location of the grid. Use the up arrow, $\uparrow$, to access previous commands; to access the previous command starting with 'x=' type 'x=' on the command line and then hit $\uparrow$. Now you can use the back $(\leftarrow)$ and right $(\rightarrow)$ to move through the previous command. Typing a character will insert it; backspace removes it. Call up the previous command defining the grid for `x` and modify it so that `x` is now an array corresponding to coordinates beginning at -5, extending to +5, and with a grid spacing (Δx) of .1. Recreate the meshgrid for `X`, `Y`, redefine the matrix `g` whose values are given by the rule above, and plot the result using

```
surf(X,Y,g),shading flat
```

The **shading flat** option removes the grid lines and shades each portion of the graph as a flat polygon. You can change the color mappings by trying things like **colormap hot**, and on a computer screen things are more striking if you issue the command **whitebg('k')** (which makes the background color black, or `'k'`). To see a particular slice of the graph, that is a section in either the `x` or `y` direction, you can plot a particular row or column of `g`. For example, to plot a cross section in the x direction, with $y = -2.5$, type

```
figure(2), plot(x, g(1,:) )
```

The colon (:) in the second argment of g says 'all available indices,' and in this case '`g(1,:)`' means 'From the matrix `g` take the first row and all columns,' which is, of course the vector containing samples taken from `g` in the `x` direction at the smallest `y` value (which has index 1). Use **hold on, hold off** to plot some other cross sections of the function until you feel comfortable with which indices correspond to which variables.

Now let's plot contours and densities in another window. Try

```
figure(3), pcolor(X,Y,g), shading flat, colormap hot
```

This will create a new figure, put a density plot of `g` in it, and color-code the densities in such a way that lower values are dark (cool) and higher densities are white (hot). It is possible to super-pose contours on this graph:

```
hold on, contour(X,Y,g,'b'), hold off
```

This will place 20 aesthetically chosen contours in blue on the color density plot.

EXERCISE 3: The function

$$u(x,t) = \frac{1}{2}\left[\exp\left(-(x-2t)^2\right) + \exp\left(-(x+2t)^2\right)\right]$$

is a solution to the wave equation, $u_{tt} = 4u_{xx}$, which models, among other things, waves on the surface of a canal, in this case moving at speed 2 in either direction. Define a meshgrid in t and x which will allow you to visualize this solution for times between 0 and 4. Use surface plots, density plots, and contours to visualize the behavior of this solution. Also, in a third figure plot a 'waterfall' diagram, that is several slices in constant time, varying `x`, which will illustrate the evolution of the function. Use times 0,1,2,3,4 and color-code the slices in some way so that you know which slice is which.

1.2.5 Basic Logical Functions

One of the nice features of MATLAB is the existence of logical matrices, which are just arrays with elements that are either zero or one depending on whether a statement is true or false. Let's return to `x` and `f` given in the first section above. Re-input these into MATLAB (remember ↑!) and now let's manipulate using logical matrices. For example, suppose that we would like to plot *only* the positive portions of f. Define

```
fplus=( f>=0 );
```

The array `fplus` is exactly as long as `f` (that is 101 elements) but contains zeros where f<0 and ones where f≥0. To plot `f` in yellow and only its postive part in little red circles,

```
plot(x, f, 'y', x, fplus.*f, 'ro')
```

By multiplying `f` and fplus element by element we have created a new entity which is the positive part of `f` where `f` is positive and zero where `f` is negative. If we wanted to show the negative part of `f` in little blue triangles on the same graph, we could

```
hold on, plot(x, (1-fplus).*f,'b^'),hold off
```

These logical matrices can also be used to create piece-wise functions, which we will see next.

Suppose you wanted to make a function p which was Gaussian for two standard deviations (2σ), and then uniform (with value $p(2\sigma)$) for two standard deviations, and then zero, and then that you wanted to re-normalize it so that the resulting function still represented a probability density function (pdf). Pick a standard deviation of .5; a normal pdf with standard deviation .5 is given by

$$N = \sqrt{\frac{2}{\pi}} e^{-2x^2}.$$

First let's define the space and normal vectors:

```
x = 6/100*[-50:50];
N = sqrt(2/pi)*exp(-2*x.^2);
```

Now we need logical matrices for the different portions of the pdf we are building. These we will call '`norm`' and '`uni`,'

```
norm=(abs(x)<=1);
uni=(abs(x)>1 & abs(x)<=2);
```

The MATLAB command **abs(x)** returns the absolute value of `x`; the statement '`(abs(x)<=1)`' tests whether or not the absolute value of `x` is smaller or equal to 1, returning the value '1' if it is, and '0' if it is not. The arrays `norm` and `uni` will thus contain only ones where the pdf will be normal and then uniform, respectively. Now we can create a vector, `pdf`, which will have the proper shape but will not have unit integral:

```
pdf=N.*norm + sqrt(2/pi)*exp(-2)*uni;
```

You should plot this to see that it is correct. Finally we must normalize. We have defined a grid with spacing 6/100=.06, so a quick approximation of the actual integral would be

```
mass=.06*sum(pdf)
```

The command **sum** adds up all the elements of a vector input, so that the above is equivalent to a simple Reimann-sum approximation to the integral. A better approximation would come from the trapezoid rule

```
mass=.06*trapz(pdf)
```

which sums the *average* value in each cell. Now we can normalize,

```
pdf=pdf/mass;
```

Now plot this. You can use up-arrows to just find the command you used last time, or you can compare it to the previous (unnormalized) function by using **hold on** and **hold off**.

EXERCISE 4: A function which we will see again in dispersal is the steady-state solution to the diffusion-loss model with a localized source at the origin,

$$u_t = Du_{xx} - \lambda u + \alpha\delta(x),$$

the steady-state ($t \to \infty$) solution of which is

$$F(x) = \frac{\alpha}{2\sqrt{D\lambda}} \exp\left[-\sqrt{\frac{\lambda}{D}}|x|\right].$$

Unfortunately this predicts a small, but perceptible presence of a diffusing particle infinitely far away from the origin. One way to address this would be to chop off the function F at some realistic point and then normalize the resulting function. Let $\alpha = 1, \lambda = 4$ and $D = 1$ and truncate the function F at 50, 25, and 10 percent of its peak value, then re-normalize so that the resulting function is a pdf. On a single graph, compare these three pdfs to a (normalized) version of the original function.

1.2.6 Learning about Matrix Manipulation and Fourier Transforms using the Demos

Now that you have a MATLAB basis, you might find it useful to tour through some of MATLAB 's capabilities using **demos**. The demos are a set of GUI slide shows which illustrate various MATLAB functions. So, on the command line type

```
demo
```

and a window should pop up on screen with a variety of MATLAB subjects. To get comfortable with the demos and see some of how MATLAB handles linear algebra click on **Matlab** and then **Mathematics** in the left hand window, followed by **Basic matrix operations** in the right hand window. If you now click on “Run in the Command Window” a slideshow begins running in the command window to illustrate various MATLAB matrix operations, which are useful to know. You can actually fool around with the commands appearing in the slide show in your command window to see how they work, which is a good way to learn.

EXERCISE 5: The Nicholson-Bailey model for host-parasite interactions is

$$\begin{aligned} N_{t+1} &= \lambda N_t e^{-aP_t} &\overset{\text{def}}{=} F(N_t, P_t), \\ P_{t+1} &= cN_t \left(1 - e^{-aP_t}\right) &\overset{\text{def}}{=} G(N_t, P_t). \end{aligned}$$

The steady state corresponding to coexistence of host and parasite in this model is given by

$$P^* = \frac{\ln(\lambda)}{a} \quad N^* = \frac{\lambda \ln(\lambda)}{ac(\lambda - 1)}.$$

The Jacobian matrix (found by hand, not MATLAB !) is

$$\begin{pmatrix} F_N(P^*, N^*) & F_P(P^*, N^*) \\ G_N(P^*, N^*) & G_P(P^*, N^*) \end{pmatrix} = \begin{pmatrix} \lambda e^{-aP^*} & -aN^* \lambda e^{-aP^*} \\ c\left(1 - e^{-aP^*}\right) & acN^* e^{-aP^*} \end{pmatrix}.$$

Show that when $a = 0.068, c = 1, \lambda = 2$ (corresponding to the interaction between a greenhouse whitefly and its chalcid parasitoid) that the steady state is unstable. At the very least you will need to take the eigenvalues of a 2x2 matrix and to see if any of these eigenvalues have magnitude greater than one. Can you think of a way to plot a stability diagram in λ which will illustrate that these equilibrium populations are *never* stable?

Now let’s have a look at using the FFT (or Fast Fourier Transform) in MATLAB . To begin with, click **Using FFT in MATLAB** in the left-hand window and then **Run in Command Window** in the upper right hand corner. You can now page through slides which illustrate how one can use the FFT to analyze sunspot data. *I suggest typing*

```
help fft
```

first to get a idea of what the FFT is all about, if you have never used Fourier transforms for data analysis. We will be using FFTs a lot, so don't despair if these seem a little cryptic just now. Remember, if you want to see what any of the component commands in the demo do, you can always type **help SUBJECT** in the command window and it will almost certainly give you either no information or more information than you really need.

Congratulations! You have 'jumped in the deep end' with MATLAB *, but you didn't sink!*

2 Dispersal

2.1 Goals

During this lab students will use MATLAB to become familiar with methodology for simulating the probabilistic dispersal of organisms using FFTs (Fast Fourier Transforms). In particular, students will:

- Visualize and understand a variety of dispersal kernels in one and two dimensions.
- Be aware of the relationship between probability kernels and special solutions to partial differential equations (PDE).
- Use FFTs and convolution to implement population dispersal models.
- Use MATLAB 'M' files to simplify simulations.

2.2 Probability and Dispersal in One Space Dimension

2.2.1 Dispersal via Random Walks and the Diffusion Equation

Many of the basic probability kernels associated with population dispersal find their basis in partial differential equations. For example, the normal distribution is associated with the diffusion equation

$$u_t = Du_{xx}.$$

The diffusion equation, among other things, models the probability (u) of finding an individual near any spatial locus (x) at some time t, given that the individual is moving by taking randomly chosen steps of size λ to the left or right at a rate of one step per time interval τ. The step size and time interval occur in the diffusion equation as

$$D = \frac{\lambda^2}{2\tau}.$$

For the dispersal of a population, the important solution to the diffusion equation is the one which begins with a perfectly localized individual at the point $x = 0$ (in math we know this as the *Dirac delta function*, $\delta(x)$), so that the initial condition is $u(x, 0) = \delta(x)$. The corresponding solution, K_D, is called the fundamental solution and has the form

$$K_D(x,t) = \frac{1}{\sqrt{4\pi Dt}} \exp\left[\frac{-x^2}{4Dt}\right].$$

This function can be interpreted as the pdf associated with the location of an individual at time t, moving under random walks, which was initially located at $x = 0$.

Now, suppose a population of organisms is dispersed over space with an initial density of $P(x, 0)$. One can think of the number of individuals located in a small interval of size dy at a location y as $P(y, 0)\ dy$. These indidviduals disperse randomly according to the distribution K_D; thus, their probabilistic location at a later time will be

$$P(y,0)\ K_D(x-y,t)\ dy,$$

where the dispersal kernel has had its argument shifted so that it is now centered at the original locus of individuals ($x = y$). The total population after a time t would then be the sum over all such infinitesimal intervals containing populations:

$$P(x,t) = \int_{-\infty}^{\infty} P(y,0)\ K_D(x-y,t)\ dy \overset{\text{def}}{=} P(x,0) * K(x,t).$$

This latter operation is called the *convolution* and is defined by the preceding integral.

2.2.2 Convolutions and Fourier Transforms

The Fourier Transform of a function defined on an interval from $-\infty$ to ∞ is

$$\hat{f}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ikx}\, dx.$$

The transformed function $\hat{f}(k)$ can be thought of as the amount of 'energy' the function has stored in the 'wave' $e^{ikx} = \cos(kx) + i\sin(kx)$. A convenient property of the Fourier Tranform (FT) is that the FT of the convolution is the product of the FTs,

$$\widehat{f * g}(k) = \frac{1}{2\pi} \hat{f}(k)\hat{g}(k).$$

Consequently, a slick analytic way to evaluate a convolution without actually doing the nasty integral is:

1. Calculate the FT of f and g.
2. In wave space, evaluate the product of $\hat{f}(k)$ and $\hat{g}(k)$.
3. $\widehat{f * g}(k) = \frac{1}{2\pi} \hat{f}(k)\hat{g}(k)$.
4. Invert the transform of $\widehat{f * g}$ to obtain $f * g$.

A significant difficulty is that FTs are not all that easy to calculate or invert. Fortunately, there is a very simple computational routine, called the Fast Fourier Transform (FFT) which will perform these operations numerically.

2.2.3 Numerical FFT

The numerical (or Discrete) Fourier Transform (DFT) is defined with summations instead of integrals,

$$\hat{f}_k = \sum_{j=-N/2}^{N/2} f(j\Delta x) e^{2\pi i k j \Delta x},$$

where the function f is thought of as being defined on the interval from $x = 0$ to $x = 1$, chopped up into N chunks of length Δx. If N is a power of two, (that is $N = 2^m = 2, 4, 8, 16, 32, 64, 128, \cdots$) there is an efficient numerical algorithm for calculating the DFT, called the FFT. One curiosity of the FFT is that (depending on the implementation) it generates a factor of N in the output, that is, for input $f(j\Delta x)$ it gives output $N\hat{f}_k$ instead of simply $\hat{f}_k$. This happens because one of the computational efficiencies of the FFT is that all the operations involve rational numbers with denominator N. MATLAB therefore multiplies by N to take advantage of integer arithmetic. Consequently, the FFT of a convolution of two functions defined on the interval $x \in [0, 1]$ is

$$\widehat{f * g}_k = \frac{1}{N} \hat{f}_k \hat{g}_k.$$

Suppose now we have a population function p defined on a periodic interval $x \in [-L, L]$, and a dispersal function, f defined on the same interval. A blueprint for calculating the dispersed population $f * p$ using the FFT follows:

- Use FFT to calculate $\hat{f}_k$ and $\hat{p}_k$.
- Calculate $\widehat{f * p}_k = \frac{2L}{N} \hat{f}_k \hat{p}_k$. The extra factor of $2L$ allows for a change in the length of the interval from 1 to $2L$. As a note of interest, the fraction $\frac{2L}{N}$ is equal to the discrete step size in space, Δx.
- Invert the FFT to arrive at $f * p(x)$.

An added wrinkle is that, although it is theoretically natural to center the interval at $x = 0$, in discrete terms this is unnatural. In general a *shift* has to be performed on the discrete version of p so that the value of p at $x = 0$ is associated with the first element of a vector, as in the following diagram:

2.2.4 Matlab Implementation

We can use FFTs in MATLAB to calculate convolutions rapidly. Let's begin by illustrating the procedure for a population initially localized uniformly between -1 and 1, dispersing over time into a much larger domain. The FFT assumes periodicity, so if we define a space interval between -10 and 10 with 256 nodes (remember the FFT is much more efficient for computations involving 2^m elements), we need to take into account that the node corresponding to 10 must be left off (the FFT assumes that whatever data might be specified there is identical with the data at $x = -10$). So, to define this space:

```
xlr=10; np=256; dx=2*xlr/np;
x=dx*[0:np-1]-xlr;          % OR, equivalently
x=linspace(-xlr,xlr-dx,np);
```

The independent variable is now defined. Let's also define the normal dispersal kernel and the initial population:

```
p0 = (abs(x) <= 1);
t=1; D=1;
K=1/sqrt(4*pi*D*t)*exp(-x.^2/(4*D*t));
```

With the relevant functions defined we can now take the Fourier transform

```
fp0=fft(p0);
fK=fft(K);
```

Using the fact that the transform of the convolution is the product of the transforms, we can now evaluate the convolution:

```
fp1=fK.*fp0;
p1=dx*fftshift( ifft( fp1 ) );
```

The vector `p1` now contains the dispersed population after 1 time unit, but there are some weird things in the command that need explaining. First off, the command **fftshift** basically chops a vector in half and interchanges the first and second halves as blocks. In this context it is necessary because the MATLAB FFT is built on the assumption that functions are defined on the interval from 0 to 1, as opposed to centered around 0. Secondly, the factor of `dx`. Firstly, for numerical efficiency the FFT multiplies a vector by `np`, and the inverse FFT divides by that factor. Since we have implemented a product of FFTs, we need to divide out a factor of `np`. Secondly, numerical FFTs assume that functions are defined on an interval of length 1, but our function is defined on an interval of length `2*xlr`, and so the FFT must be scaled by the interval length. Together, these two conditions are equivalent to multiplying by the step size, `dx=2*xlr/np`, which is convenient from our perspective.

Now let's see what we have done:

```
plot(x, p0,'r', x, p1,'y')
```

You should also get a warning about imaginary numbers being ignored. This occurs because the FFT requires complex numbers, and numerical rounding creates very small imaginary components, even for perfectly real data fields. The best way to get around this is to take the **real** part after convolution, which we will do in the future. Now it is possible to keep iterating the random dispersal of the population. We can get the population after the next time step using a single long, cryptic MATLAB command:

```
p2=dx*real( fftshift( ifft( fft( p1 ).*fK ) ) );
```

Use **hold on** to plot this against the previous results. Does it look reasonable? Use the up-arrow (↑) and edit the previous command, plot, and see the population after dispersal over three time units.

EXERCISE 6: Use MATLAB to define the population density function

$$P(x,0) = \begin{cases} 2-2|x|, & -1 \le x \le 1, \\ 0, & |x| > 1, \end{cases}$$

on the interval from -15 to 15. In a single window plot random dispersal of this population after times of .5, 1, 2, and 4 with diffusion constant $D = 2$. (Using ↑ here will simplify your life!) Use **trapz** to approximate the integral of the population (total number of individuals) and convince yourself that the number of individuals is being conserved.

2.2.5 Dispersal and Settling with a Localized Source

Another interesting dispersal model involves assuming random motion for propagules (seeds, spores, pollen, motile larvae) while they are in the medium and then settling onto a substrate at some rate λ. If propagules are initially released at the point $x = 0$ the PDE modelling the density of propagules in the medium is

$$u_t = D\, u_{xx} - au, \quad u(x,0) = \delta(x).$$

The deposition of these particles on the ground is tracked by the variable v, which satisfies

$$v_t = av, \quad v(x,0) = 0.$$

The long-time ($t \to \infty$) solution for v is a dispersal kernel, and can be written (following Neubert et. al. 1995)

$$K_S(x) = \sqrt{\frac{a}{4D}} \exp\left[-\sqrt{\frac{a}{D}}\,|x|\right].$$

The following commands plot this function on the previously defined `x` interval:

```
D=1; a=2;
KS=sqrt(.25*a/D)*exp(-sqrt(a/D)*abs(x));
plot(x,KS,'r')
```

Now try plotting KS on the same axes, varying the settling rate, a, to get an idea of how it behaves.

EXERCISE 7: For a uniform distribution of sources between -1 and 1 investigate the dispersal of propagules according to the probability distribution K_S given above. On the same set of axes plot the effect of changing a on the dispersal of the population.

2.2.6 Ballistic Dispersal

A model for ballistic dispersal of spores from ground level in random directions is given by Neubert, Kot and Lewis, 1995, by

$$K_B = \begin{cases} \frac{1}{\pi\sqrt{\frac{c^4}{g^2} - x^2}}, & |x| \le \frac{c^2}{g}, \\ 0, & |x| > \frac{c^2}{g}. \end{cases}$$

Here c is the speed at which spores are ejected and g is the gravitational constant. Under the assumption that every spore germinates and the sporulating adults die each generation, the spread of a population of fungi is modelled by

$$P_{n+1}(x) = K_b * P_n(x),$$

where n denotes the generation number and $P_0(x)$ denotes the initial distribution of the colony.

We could illustrate the spread of the colony using the commands we have already mastered in MATLAB , but this is a good place to use an ‘m’ file, which is a MATLAB script which runs a sequence of commands. In the *File* menu of the MATLAB command window click on *New* and then *M-file*. An editor/debugger window will pop up, in which you can type in a series of commands, save it as an M-file, and then execute the whole sequence by referring to the M-file. Below appears the text for an M-file modelling this dispersal.

```
%
%   BALLSPORE
%
%   a Matlab M-file which models ballistic dispersal of fungi in 1-D
%

pold = pnew;    % define initial condition, this step --
                %     USER MUST DEFINE THE FIRST pnew
                %     BEFORE RUNNING THIS SCRIPT

KB=1./(pi*sqrt(c^4/g^2-x.^2)).*(abs(x) <= c^2/g);
                % define dispersal kernel, taking care that it is zero in
                %   the right places

KB=KB/trapz(dx*KB);
                % this normalizes the probability kernel.  Because of
                %   the singularities at the ends this is an important
                %   step -- otherwise there will be net loss

fKB=dx*fft(KB);
                % Take FFT of dispersal kernel, multiplying by dx to save
                %   that step later

fpnew = fKB.*fft(pold);
                % FFT of new dispersed population
pnew = real(fftshift(ifft(fpnew)));
                % new dispersed population

hold on, plot(x,pnew,'r'), hold off
                % plot results on current figure
```

When you have this all typed in to the editor, save the file as **ballspore.m** and click on the command window to activate it. *Make sure that the directory you are in in the command window is the same as the directory you save the program* `ballspore.m` *in!* Now we must set the initial space up, the parameters, and the initial colony:

```
     g=980;                           % cm/s^2 for gravity constant
     c=50;                            % cm/s speed of ejection
     xlr=15; np=256; dx=2*xlr/np;
```

```
x=-xlr+dx*[0:np-1];         % define independent variable
pnew=(abs(x) <= 1);         % initial colony localized between +/-1
```

Now we can plot the initial population and call **ballspore**, which will plot subsequent dispersed populations on the same graph. So, in the command window type:

```
plot(x, pnew,'b')
ballspore
```

Now, you can see the evolution of the dispersing population by using ↑ to bring back the `ballspore` command and then carriage return to execute the command.

2.2.7 Movement at Constant Speed from a Source and Settling

If propagules (density u) move at constant speed to the left and right away from a source and settle at some rate $h(t)$, the precipitated individuals (density v) satisfy the following system of PDE (from Neubert et. al., 1995):

$$u_t = -c\,\text{sign}(x)\,u_x - h(t)\,u, \qquad v_t = h(t)\,u.$$

If these equations are solved with initial conditions $u(x,0) = \delta(x), v(x,0) = 0$ corresponding to a unit release of propagules precisely at the origin, the final distribution of settled individuals (K_A) satisfies

$$\frac{1}{2c}h\left(\frac{|x|}{c}\right)\exp\left[-\int_0^{\frac{|x|}{c}} h(s)\,ds\right].$$

This solution differs from K_S primarily in that there is no diffusion process operating. Choosing $h(t) = 3a^3t^2$ gives an interesting bi-modal solution

$$K_A = \frac{3a^3|x|^2}{2c^3}\exp\left[-\left(\frac{a|x|}{c}\right)^3\right].$$

(see Neubert et. al. for other choices for h and their consequences). Try plotting this kernel for various choices of c and see how it behaves.

EXERCISE 8: Under the assumption that every settled propagule becomes a propagule-producing adult, write an M-file in MATLAB that will allow you to investigate the multi-generational dispersal of an initially-localized population with dispersal probability given by K_A. How would you modify this code to reflect mortality or non-germination – say only 50% of dispersed individuals survive to become adults? Could you modify it to have more than one generation producing propagules?

2.3 Probability and Dispersal in Two Space Dimensions

Now we have more than enough understanding to move on to two-dimensional dispersal. Mainly there is nothing more complicated than in one dimension, except that it can be computationally much more time consuming and more difficult to visualize. But the computational techniques are identical in MATLAB , except that a few commands (mainly the FFT commands) need to be changed to 2-D versions.

2.3.1 Dispersal via Random Walks in Two Dimensions

As in the case of one dimensional random dispersal, dispersal generated by random walks in two dimensions is modelled by the diffusion equation,

$$u_t = D\nabla^2 u = D(u_{xx} + u_{yy}).$$

The fundamental solution corresponding to initially localized data $u(x, y, 0) = \delta(x)\delta(y)$ is given by

$$K_{2D}(x, y, t) = \frac{1}{4\pi Dt} \exp\left[-\frac{x^2 + y^2}{4Dt}\right].$$

The following MATLAB commands define space and the dispersal kernel K2D:

```
xlr=10; ylr=10; np=128; dx=2*xlr/np; dy=2*ylr/np;
x=linspace(-xlr,xlr-dx,np);
y=linspace(-ylr,ylr-dy,np);
[X,Y]=meshgrid(x,y);
t=1; D=1;
K2D=1/(4*pi*D*t)*exp(-(X.^2+Y.^2)./(4*D*t));
```

You may wish to take a second and visualize this using **surf** or **pcolor** and **shading flat** or **shading interp**. Now, to see how a population initially localized in the rectangle $(-1 \le y \le 1) \times (-3 \le x \le 3)$ disperses, use:

```
p0 = ( abs(X)<=3 & abs(Y)<= 1 );
fp0 = fft2(p0);
fK2D = dx*dy*fft2( K2D );
fp1 = fp0.*fK2D;
p1 = real( fftshift( ifft2( fp1 ) ) );
```

Some things to observe: the overall format for implementing dispersal is the same as in one dimension, with the exceptions that `fft` and `ifft` are replaced by `fft2` and `ifft2`; there is now a multiplication by both `dx` and `dy` to implement the convolution. To visualize how the initial field relates to the dispersed field we will use a combination of graphics – one choice would be contours for the initial field and density plots for the dispersed field, as in:

```
pcolor(X,Y,p1),shading flat, hold on, contour(X,Y,p0,'m'), hold off
```

You may want to further use the options **axis square** and **colormap hot**. The `hot` colormap, in particular, is useful in interpreting density plots, since it gives a clear indication of high and low values. To see which colors correspond to which values, you may also use the command **colorbar**, which will add a color scale with associated scalar values. You might also want to try a combination surface/contour plot, as in

```
surf(X,Y,p1+1), shading flat
hold on, contour(X,Y,p0,'m'), caxis('manual'), hold off
```

The `caxis` command controls how colors are scaled; including the `'manual'` option keeps the contour plot from messing up the beautiful colormap of **surf**. For most applications I find it easiest to use combinations of density and contour plots.

<u>EXERCISE 9</u>: Disperse this population for one more time step, saving `p1` and creating a new, twice-dispersed population, `p2`. Plot all three populations on a single graph using contours in two different colors

for `p0` and `p1` and a hot-colored density plot for p2. Use `dx*dy*trapz(trapz(p2))` to estimate the double integral of `p2`, and then compare that to the previous two dispersal slices to make sure that the population is being conserved.

2.3.2 Modal Dispersal in Two Dimensions

Well, there were many, many different things to do in one dimension, and as you might guess there are infinitely many more in two dimensions. We will discover more of these as we go along, but as a parting shot let's model a combination of advection and diffusion in two dimensions. Suppose a population is performing random walks in a medium which moves in a particular direction with velocities u and v in the x and y directions, respectively. Without belaboring the details, it can be shown that an individual initially localized at the origin disperses with probability

$$K_{2A} = (x, y, t) = \frac{1}{4\pi Dt} \exp\left[-\frac{(x-ut)^2 + (y-vt)^2}{4Dt}\right].$$

<u>EXERCISE 10</u>: Investigate the advective/diffusive dispersal of a population which is initially localized, according to the probability kernel K_{2A}. Pick $D = .25$, $u = 2, v = 2$, and use the command

```
p0=( abs(X+8) <= .5 & abs(Y+8) <= .5);
```

to specify the initial population. Define the dispersal kernel K_{2A} in MATLAB using the previous grid, and using a sequence of commands on a single line disperse this population over one unit of time. Using a combination of contours and density plots show the population's progress over three time units of dispersal. (*Alternatively,* you may want to write a single M-file to do this.)

<u>EXERCISE 11</u>: Build an M-file which will allow you to investigate the behavior of a population in an advection/diffusion setting with *random winds*. That is, over each period of dispersal ('day') the wind speed is constant, but from day to day the direction will vary randomly. *Hint:* In MATLAB the command for generating uniformly distributed random numbers is **rand**. Visualize a couple of dispersal steps for a population localized initially uniformly in a disk of radius 1 centered at (0,0). Would it be hard to modify your program to account for random changes in wind speed as well?

2.3.3 Turchin's Model for Prey-Taxis

Another characteristic of moving, living things is that they move for a reason, generally to get somewhere or avoid something. An example would be predators or parasites searching for prey/hosts. Even the simplest of beings is likely to slow down, that is, to take shorter steps in its random searching motions, when it is in the presence of prey items. Let $V(x, y)$ be the density of 'victims' in space, and let $P(x, y, t)$ be the density of searchers (predators/parasites). Let D_2 be the diffusivity of P in the absence of victims and $D_1 < D_2$ the diffusivity in the presence of victims. Varying *motility* of the searcher can be modelled (following Turchin, Reeve, Cronin and Wilkens, 1997) by

$$\mu(V) = \frac{D_2(V+d)}{d + V\frac{D_2}{D_1}} = D_2\left(1 - \underbrace{(D_2 - D_1)\frac{V}{D_2 V + dD_1}}_{\hat{\mu}}\right),$$

where d is a saturation parameter describing how rapidly motility changes with density of victims. The function $\hat{\mu}$ can be intepreted as the prey-based perturbation to the 'normal' motility of predators in search mode. A PDE model for the redistribution of searchers in response to the density of victims is

$$P_t = \nabla^2 \left[\mu(V)P\right].$$

The idea is that in regions with prey predators slow down (to eat the prey, or at least to hug them and pet them and stroke their fur the wrong way), while in regions without prey they speed up (searching). Thus, when a distribution of predators encounters a distribution of prey the predators tend to 'pile up' as they slow down and have more intensive interactions with their victims. In principle this equation is always soluble, but there is no *a priori* form of solution that works for all choices of V. However, an *approximate* solution with initial condition $P_0(x,y)$ is given by:

$$P(x,y,t) = \frac{1}{4\pi\mu(V)t} \cdot \left[\left((1-\mu(\hat{V}))P_0(x,y)\right) * \exp\left(-\frac{x^2+y^2}{4D_2 t}\right)\right].$$

Remember that * denotes the two-dimensional spatial convolution; in this case the entire quantity $(1 - \mu(\hat{V}))P_0(x,y)$ is convolved with a diffusion kernel with diffusion constant D_2.

This approximation to solutions of the PDE improves when $\hat{\mu} << D_2$ (as the diffusivities become more alike or V is either much smaller than or much greater than $\frac{dD_1}{D_2}$). However, it always has the correct quantitative behavior, even when its mathematical properties as an approximate solution are unreliable. For our purposes, as a *model* for victim-taxis, the approximation above can serve perfectly well! We must separate the elements which give the random character of diffusivity from the elements defining the spatial structure of changing motility in the presence of prey. The approach will be to:

1. pre-multiply by the differential motility, $(1 - \mu(\hat{V})$, which creates a bias for the resulting 'motility potential' to diffuse into areas where $\mu(V)$ is small,
2. convolve with a random motion kernel with diffusivity D_2,
3. divide the dispersed result by the differential motility to convert realized 'motility potential' back to an actual spatial structure of dispersed predators
4. normalize to ensure that predators have neither been created nor destroyed (because the approximate solution does not conserve the number of predators precisely).

Below appears a MATLAB m-file which implements this model for prey-taxis, assuming an initial population of prey distributed in a Gaussian oval centered at $x = 5 = y$,

$$V(x,y) = 100 \exp\left[-\frac{(x-5)^2}{4} - (y-5)^2\right],$$

and a population of predators initially distributed uniformly in a circle of radius 2. In particular, notice the 'normalization' step which keeps the predator population from growing or shrinking, and the separation between the purely random motion (evaluated with convolution) and the pre/post adjustment by a factor of `1-muhat`, which gives spatial structure to the taxis.

```
%
%    Testing out Turchin-taxis
%
%    Set up space:
%
   np=128;
   xl=10; dx=2*xl/np;
   x=linspace(-xl,xl-dx,np);
   [X,Y]=meshgrid(x,x);

%   Define parameters

   D2=4; D1=1.5;                        % the motilities
   d=1; t=10;                           % saturation parameter, time
   V=100*exp(-(X-5).^2./4-(Y-5).^2); % these are the victims:

%   muhat is (negative) perturbation to the basic motility (D2):
   muhat=( D2 - D1 ) * V./( d*D1 + D2*V );

%   Set initial population of predators

   P=( (X.^2+Y.^2) <= 4 );
   numP=dx^2*trapz( trapz( P ) );    % total number predators

%   Define dispersal kernel with diffusivity D2

   K=1/(4*pi*D2*t)*exp(-(X.^2+Y.^2)/(4*D2*t));
   fK=dx.^2*fft2(K);

%   Disperse motility potential MP=(1-muhat)*P
   MP=(1-muhat).*P;
   fMP=fft2(MP);
   MPt=real(fftshift(ifft2(fMP.*fK)));

% convert back to actual population from realized motility potential
   Pt=MPt./(1-muhat);

%   Normalize the dispersed population

   numPt=dx^2*trapz( trapz( Pt ));  % the number we now have
   Pt=numP/numPt*Pt;                % convert to number we should have

%   Plot the results in comparison with the victims

   pcolor(X,Y,Pt), shading interp, colormap hot, axis square
   hold on, contour(X,Y,V,'b'),  caxis('manual'), hold off
```

One difficulty with this sort of approach is that it is not ‘linear’ in t, the time step. That is, taking ten steps with $t = 10$ could be slightly different than taking five steps with $t = 2$ or two steps with $t = 5$. Similarly, taking one step with $D_2 = 8$ may not be exactly equivalent to taking four steps with $D_2 = 2$. However, these differences are normally small where the method works at all, and as a model for chemotaxis it has the correct qualitative behavior, no more or less valid in an absolute sense than the PDE model appearing in the paper of Turchin et. al. (1997). It is important, though, to remember that there is now an extra ‘parameter’, t, which must be specified.

EXERCISE 12: With the above MATLAB code for defining a prey-tactic dispersal kernel, determine how a population of predators, initially localized near $x = 0 = y$, redistributes in response to the population of victims given above. In two different figures plot the density response of the predators as a density field after 4 and 8 time units of dispersal. Indicate the starting point, and plot contours of the victim field to see how the taxis orients the population. Check to see that the number of searching individuals is being preserved. Can you explain these results in terms of decreased motility in the presence of they prey?

EXERCISE 13: Write an m-file script which allows you to iterate the prey-tactic model of the dispersal process above (with time steps of 1 unit) with $D_2 = 4$ in a situation with predators searching for prey infesting crop rows, as in

```
V= 10*exp(-(X-4).^2/9-(Y-4).^2/9).*(1-cos(20*pi/xl*X));
```

Congratulations! You have learned more about probabilistic dispersal and convolutions in MATLAB *than most people are learn in a life time!*

3 Implementing an Integro-Difference Model

3.1 Goals

In this lab we will learn the philosophy and some of the practical MATLAB pitfalls associated with implementing an Integro-Difference Equation (IDE) model. An IDE is one natural way to add spatial dynamics to a temporally discrete model. In this lab you will:

- Learn how MATLAB implements movies and loops.
- Couple dispersal models with discrete-time population dynamics.
- Learn some ways to implement environmental heterogeneity.
- Begin to see some of the unique behaviors associated with spatio-temporal dynamics.

3.2 Enhanced Nicholson-Bailey Model

For purposes of demonstration we will look at version of the Nicholson-Bailey discrete model for the interaction between host and parasite species. A nice, detailed account is presented in *Mathematical Models in Biology*, pp. 79–89, by Leah Edelstein-Keshet. The purely temporal model tracks the population of the host species, H_t, and the parasite species, P_t as a function of discrete time, t. The Enhanced Nicholson-Bailey model (ENB) maps current population levels to future population levels according to

$$H_{t+1} = \quad f(H_t, P_t) \quad = H_t \exp\left[r\left(1 - \frac{H_t}{K}\right) - aP_t\right], \tag{1}$$

$$P_{t+1} = \quad g(H_t, P_t) \quad = nH_t\left(1 - \exp[-aP_t]\right). \tag{2}$$

The right-hand-side of these equations can be viewed as a product of probabilities of encounter related to the 'Law' of Mass Action, that is, the probability of an encounter between host and parasite is proportional to $H_t \cdot P_t$. The probability of a host encountering parasites twice is $H_t \cdot P_t^2$, and the probability of encountering n parasites is $H_t \cdot P_t^n$. If a is the per parasite probability of parasitization, the probability of a host being parasitizing at least once is

$$\underbrace{aH_tP_t}_{\text{1 encounter}} + \underbrace{\frac{1}{2}a^2H_tP_t^2}_{\text{2 encounters}} + \underbrace{\frac{1}{6}a^3H_tP_t^3}_{\text{3 encounters}} + \cdots = H_t\left[1 - e^{-aP_t}\right].$$

The various fractions $\frac{1}{n!}$ occur because the *order* of the encounters does not matter. For example, there are $6 = 3!$ ways for three different parasites to encounter a particular host, but if the order of these encounters does not matter we need to factor out that 3! to get the probability of three order-independent encounters. Consequently, the right-hand-side (RHS) of 2 can be thought of as the expected number of hosts which have been parasitized at least once times the number of fuzzy baby parasites (n) produced by each successfully infected host. The product $H_t \exp[-aP_t]$ on the RHS of equation (1) is the expected number of *non-parasitized* hosts, which then reproduce with fecundity based on the Ricker map, $\exp\left[r\left(1 - \frac{H_t}{K}\right)\right]$.

There are three fixed points for ENB: $(H, P) = (0, 0), (K, 0)$ and a mixed population (H^*, P^*) which must be calculated using a nonlinear root finder on the equations

$$H^* = f(H^*, P^*) \quad \text{and} \quad P^* = g(H^*, P^*).$$

This mixed population is stable for an intermediate range of fecundity (r) and parasite effectiveness (a). When a is sufficiently small the mixed population is unstable and the parasites go extinct ($H_t \to K$). When a is sufficiently large the mixed population is also unstable, but in this case there can be stable limit cycles, oscillations leading to extinction, or period-doubling bifurcations leading to chaos.To see discrete population dynamics in MATLAB you will need to implement a **for** loop. For example, to see behavior in the ENB you can implement the following M-file:

```
%
%   ENB -- a small script for implementing the
%          Enhanced Nicholson-Bailey model
%

   n=1; a=4; r=1.75; k=1;        % set parameters for ENB
   ngens=50;                     % set number of generations
   h=zeros(1,ngens); p=h;        % initialize population vectors

   p(1)=.1; h(1)=.9;             % set initial conditions

   for t=1:ngens-1               % iterate for next ngens-1 generations

      h(t+1)=h(t)*exp( r*(1 - h(t)/k) - a*p(t) );
      p(t+1)=n*h(t)*( 1 - exp(- a*p(t) ));

   end                           % of iteration over generations

   plot(h,'b'),hold on, plot(p,'r'),hold off
   legend('host','parasite')
```

Notice the `for` and `end` statements above, which define a loop in the MATLAB script. Try running this program for increasingly large r values and see how the dynamics change.

EXERCISE 14: The (un-enhanced) Nicholson-Bailey model for host-parasite interactions is

$$\begin{aligned} H_{t+1} &= \lambda H_t e^{-aP_t}, \\ P_{t+1} &= cH_t\left(1 - e^{-aP_t}\right). \end{aligned}$$

Earlier (lab 1) we examined the stability of the single non-zero fixed point for this model and showed that it is always unstable. Write a MATLAB M-file which will allow you to set initial conditions exterior to the program and then iterate for some number of generations. (*Hint:* you may wish to modify the RHS of these equations to remove the possibility of negative values!) Additionally, plot the populations *against one another* (phase space) as well as against time (state space). The parameters $a = 0.068, c = 1, \lambda = 2$ correspond to the interaction between a greenhouse whitefly and its chalcid parasitoid. Now you have visualized the well-known instability of what seems like a pretty reasonable model. Nicholson (1957) proposed that the instability and subsequent death of both species would be moderated in nature by continual re-invasion of patches.

3.3 Matlab Implementation of a Simple IDE

Probably the simplest possible integro-difference equation is one which incorporates a single step of dispersal followed by a single step of linear, Malthusian growth at a per-capita rate of r:

$$\begin{aligned} P_{n+\frac{1}{2}} &= K * P_n \\ P_{n+1} &= rP_{n+\frac{1}{2}}. \end{aligned}$$

The order of these two steps (reproduction/dispersal or dispersal/reproduction) is not important, mathematically, although it can have significant biological repercussions (see Anderson (1991) for a discussion in the context of plants). One may think of this as a model for an insect whose adult females disperse to new locations (with resulting density $P_{n+\frac{1}{2}}$) and then lay eggs with a net fecundity of r (including number of eggs laid per female, fraction of eggs which hatch, fraction of larvae which survive to adulthood). The following MATLAB script executes this scenario for simple drift and diffuse dispersal over several time steps.

```
%
%     MALTHUS
%
%     matlab code to evaluate dispersal due to convolution
%     of a spatial dispersal probability with a population density
%     function.
%
%     p(x)    -    population density in x
%     K(x)    -    probability of individual dispersal to location
%                       x starting at location 0.
%     xl -       maximum x coordinate (also - minimum)
%     dx -       spatial step  delta x = 2*xl/np
%     np -       number of points  in discretization for x
%                 (should be a power of 2 for efficiency of FFT)
%     c  -       center of spatial  dispersal
%     D  -       diffusion constant (per step)
%     r  -       per-capita growth constant
%

   np=128; xl=10; dx=2*xl/np; c=.4; D=.25;
   r=1.15;
   nsteps=10;
%
%     discretize space, define an initial population distribution
%
   x=-xl+[0:np-1]*dx;
   p=(abs(x+3)<=.5);
   p0=p;        %  keep track of initial population for comparison
%
%       define a dispersal kernel
%
   K=exp(-(x-c).^2/(4*D))/sqrt(4*pi*D);
%
%       normalize K to make it have integral one
%
   K=K/(dx*sum(K));
%
%     calculate the fft of K, multiplying by dx to account
%     for the additional factor of np and converting from a
%     interval length of 1 to 2*xl.  The fftshift accounts for
%     using an interval of (-xl, xl) as opposed to (0,2*xl).
%
   fK=fft( fftshift( K ) )*dx;
%
```

```
%     begin iterating the model
%
   for j=1:nsteps,
%
%     calculate the fft of p; perform the convolution
%

      fp=fft(p);
      fg=fK.*fp;
%
%     fg now contains the fourier transform of the convolution;
%     invert it, multiply by post-dispersal reproduction,
%      and take a look.
%
      g=r.*abs(ifft(fg));
      plot(x, p0,'r', x, g,'b'), axis([-xl xl 0 1.6])
      pause(.5)     % pause for .5 sec  to see the plot
%
%     update p and move to the next time step
%
      p=g;

   end     % of the iterations
```

There are some features to note about this script. The `pause(.5)` command causes the program to stop for half a second and plot. Try commenting out this line and running the program again – you will see only the end result as opposed to the intervening steps. Now try removing the parentheses, that is, replace `pause(.5)` with `pause`. This will cause the program to wait for user input before proceeding – for example, just hit the space bar in the command window and the program will advance one step. Finally, to see all the steps together, replace the `plot` line with

```
   hold on, plot(x, p0,'r', x, g,'b'), axis([-xl xl 0 1.6]), hold off
```

This should allow you to see all the iterations on the same plot, with one additional per each 'click' of the space bar.

EXERCISE 15: Imagine now that the net fecundity, r, depends on space – that is, eggs hatch and survive better in some regions or others. Modify the program above so that r is 1.3 for $|x| < 1.5$ and $r = .8$ elsewhere. (*Hint*: the MATLAB command `ones(1,np)` generates a row-vector of ones of length `np` and the logical statement `(abs(x) <= 1.5)` generates a vector which has elements 0 where $|x| > 1.5$ and 1 where $|x| \leq 1.5$). Will the population persist in this case? Now try the same scenario with with mean dispersal distance zero ($c = 0$).

3.4 An Integro-Difference Approach to Nicholson-Bailey

3.4.1 Description of IDE for Enhanced Nicholson Bailey

The idea for implementing an IDE approach to known discrete dynamics is to imagine that the behavior of the organisms can be represented in two stages:

- **Reproduction and lifecycle dynamics**. This is the portion of the populations' behavior that is already captured in the ENB, equations (1,2). For this particular model, we think of parasitized hosts as immobilized, and future hosts as eggs (also immobilized). The major change is that now H_t and P_t are population densities depending on space, $H_t(x, y)$ and $P_t(x, y)$.

- **Dispersal**. Cuddly fuzzy baby hosts and parasites disperse after birth, each according to species-specific dispersal probabilities, $K_H(x, y)$ and $K_P(x, y)$ respectively. The basic form of the dispersals for each species will be a form of dispersal kernel derived from the heat equation (for random diffusion in a plane):
$$K(x, y, T) = \frac{1}{4\pi\mu T} \exp\left[-\frac{x^2 + y^2}{4\mu T}\right].$$
Here T is the dispersal time of each species and μ is a species parameter describing the 'diffusivity,' or rate of dispersal, of each population.

The mathematical blueprint for the IDE approach to ENB is then

1. Calculate the number of baby hosts and parasites based on the last population values:
$$\begin{aligned} H_{t+1/2}(x, y) = \quad & f(H_t, P_t) & = H_t(x, y) \exp\left[r\left(1 - \frac{H_t(x, y)}{K}\right) - aP_t(x, y)\right], \\ P_{t+1/2}(x, y) = \quad & g(H_t, P_t) & = nH_t(x, y)\left(1 - \exp[-aP_t(x, y)]\right). \end{aligned}$$

2. Disperse the toddling baby organisms using convolutions as described above:
$$\begin{aligned} H_{t+1}(x, y) &= K_H * H_{t+1/2}(x, y), \\ P_{t+1}(x, y) &= K_P * P_{t+1/2}(x, y). \end{aligned}$$

Notice that the dispersal kernels may be different for host and parasite! By iterating these rules for many t we can simulate many generations of reproduction, interaction, and dispersal.

3.4.2 Matlab Implementation

Below appears psuedo-code which implements the above model in MATLAB. For interesting results parameter values are chosen where oscillatory instabilities and chaos occur.

```
%
%     NBSPACE
%
%     matlab code to iterate an enhanced Nicholson-Bailey model
%     for host and parasite, with each capable of diffusive movement.
%
%     Intermediate variables (*) are created  using a discrete
%     enhanced N-B step:
%
%            h* = h(t) exp[ r (1 - h(t)/k) - a p(t) ]
%            p* = n h(t) ( 1 - exp[- a p(t) ] )
%
%     This population is then allowed to disperse using a spatial
%     convolution, * below, (like a long step of  a heat equation):
%
%            h(t+1) = p_h(x,y) * h*
%            p(t+1) = p_p(x,y) * p*
%
%     p_h and p_p can be any probability functions representing the
%     dispersal of the two species; I have used Gaussians below.
%
%     Parameters appearing (and their interpretation):
%
%            r       reproduction rate of host  species
%            k       carrying capacity of host species
%            a       encounter rate or effectiveness of parasite on host
%            n       parasites produced by a successful infestation
%
%     Parameters used for the code:
%
%            np       number of points in x, y directions (a power of 2)
%            xl       length of domain in x and y directions
%            dx       grid spacing
%            dt       a time-step length
%            mu       followed by h and p -- diffusion parameter for each
%                            species
%            ngens    number of steps to run the code (number of
%                            generations

%
%     set parameters and spatial grids
%
  np=64; mup=.02; muh=.02; dt=.5; xl=10; dx=2*xl/np; ngens=20;
  x=linspace(-xl,xl-dx,np);
  y=x; [X,Y]=meshgrid(x,y);
```

```
%
%     set up spatial parameters
%

  n=ones(np);
  a=4*ones(np);
  r=1.75*ones(np);
  k=ones(np);

%
%     Set up stuff for movie
%
% M=moviein(ngens);          %       If you want movies

%
%     Set up initial conditions
%

  p0=.5*(1+cos(.5*pi*X/xl+pi/4*rand(np)).*sin(pi*Y/xl-pi/2*rand(np)));
  h0=.5*rand(np).*(1+cos(4*pi*sqrt(X.^2+Y.^2)/xl));

%
%     Define movement kernels for host and parasite
%

  hker=exp(-(X.^2+Y.^2)/(4*muh*dt))/(4*pi*dt*muh)*dx^2;
  Fhker=fft2(hker);       % FFT is taken because we will use it often,
                             % two factors of dx; one for each dimension

  pker=exp(-(X.^2+Y.^2)/(4*mup*dt))/(4*pi*dt*mup)*dx^2;
  Fpker=fft2(pker);

%
%     Now we are in a position to iterate for
%     a number of generations equal to ngens
%

  for j=1:ngens

%
%     Advance the life cycles using adjusted N-B model
%

     hn=h0.*exp(r.*(1-h0./k)-a.*p0);
     pn=n.*h0.*(1-exp(-a.*p0));

%
%      Do the dispersal step
%

     fhn=fft2(hn);            %     First take fft of each species
     fpn=fft2(pn);            %     First take fft of each species

%
%     Now do the convolutions.
%     The fftshift serves to center the probability functions.
%
     h0=real(fftshift(ifft2(Fhker.*fhn)));
     p0=real(fftshift(ifft2(Fpker.*fpn)));

%
%     Now plot the solution
%

     pcolor(X,Y,h0-p0), shading interp,  axis square, colorbar
     pause(.1)
% M(:,j) = getframe;       %     If you want movies

  end
%
%     Here ends the iteration
%

%
%  To play the movie,  use this command:
%  movie(M,2,8)
```

Now, if you want to run this M-file as it stands you should see basically a complicated spatial field at the end. This field is coded so that the difference between relative abundances of host and parasite can be seen; places with more abundant hosts will appear as red, more abundant parasites as blue. To see a movie you must un-comment the

```
        %       If you want movies
```

lines in the script. These movies can be memory-intensive, so don't get freaked out if your computer suddenly starts whirring and smoking! Also, movies are created in MATLAB basically by saving screen-grabs of the (normally computationally intensive) graphics. This allows the frames of the movie to be 'stacked' together and viewed rapidly, but it also means that you can not change the size of movie when it is being viewed, and if you pop up any windows which overlap with the MATLAB graphics window while you are producing a movie, MATLAB will grab whatever you popped up and keep it in the movie! The command

```
     movie(M,2,8)
```

will play the movie, `M`, 2 times at a speed of 8 frames per second. The advantage of having a movie is that you can play it again and again without re-doing the calculation. You can also save a movie to a transportable AVI file.

EXERCISE 16: Copy the above MATLAB M-file and modify it to implement an IDE for the normal, vanilla Nicholson Bailey model. Run the model with initial conditions which are:

1. Spatially homogenous (constant).
2. Spatially random (remember `rand`).

3. On a spectrum of structures (try taking a constant + sinusoidally varying perturbations and then change the relative magnitudes – as in `k=1+.9*cos(4*pi*X/xl).*cos(4*pi*Y/xl)`).

Can you verify Nicholson's hypothesis that space allows persistence of both species?

EXERCISE 17: Create a MATLAB M-file which will simulate the spatial spread of a single population satisfying (when space is not included) a logistic model:

$$P_{n+1} = P_n + rP_n(1 - P_n)$$

with random dispersal. For a particular (small) diffusion parameter, investigate the effect of increasing the intrinsic growth rate (r) of the population. What do you observe? Is this reasonable? What happens when r enters the chaotic regime?

3.5 Spatially Structured Environments

One of the nice things about IDE models implemented in MATLAB is that they can accommodate spatial structure easily. Returning to the ENB model for a moment, consider the ways in which the spatial environment might influence parameters in the model. It is possible that motility of organisms can vary with the environment – an example of this is provided above in the section on victim-taxis. Neglecting this, the environment can have a plethora of life-history effects, perhaps the largest of which is alteration in the 'carrying capacity,' K, of the environment for the host. Returning to the MATLAB code for the ENB given above, notice that when parameters are defined they are defined as *matrices* of the same size as all spatial variables. When the parameters are used in the life-history step of the IDE, the MATLAB implementation accounts for element-by-element matrix multiplication. Consequently, if we wanted to simulate the effect of a crop on the host, we could insert

```
k=.1*ones(np)+(abs(X)<=.25)+(abs(X)>=1.75 & abs(X)<=2.25)...
   +(abs(X)>=3.75 & abs(X)<=4.25)...
   +(abs(X)>=5.75 & abs(X)<=6.25)...
   +(abs(X)>=7.75 & abs(X)<=8.25)...
   +(abs(X)>=9.75 & abs(X)<=10.25);
```

in place of the existing line

```
k=ones(np)
```

(Notice the MATLAB form of 'continuation' for a long line: the ellipsis '...') This will have the effect of putting 'stripes' of high carrying-capacity crops in the environment.

EXERCISE 18: Implement the crop-structured spatial heterogeneity indicated above. Think of a way to introduce parameters so that the width of the crop stripes and their relative 'value' to the host are easy to adjust. Try some simulations – are there 'crops' and spatial structures which are more or less vulnerable to infestation by the host even in the presence of the parasite?

EXERCISE 19: Suppose that the cropping structure also influences the 'efficiency,' a, of the parasite. How should this be included, and what are its effects on the dynamics?

Although they may be tedious, artificial spatial structures are much easier to introduce than ‘natural’ ones, just because a semi-structured ‘random’ environment is very hard to put together off the top of your head. Of course, in principle one could take field measurements of a spatial arena and then scan these into density plots, but probably that is beyond the scope of our current lab experience. One alternative was suggested to me by Professor Jon Allen. The following MATLAB script, named ‘rockies.m’ generates a randomized but spatially correlated structure by choosing random phases and angles in Fourier spectrum, but using a user-specified power law for decreasing energy in the power spectrum. (If this doesn’t make sense to you, don’t sweat it; the program will work anyway). The MATLAB script requires a ‘fractal dimension’, `H`, which increases the spatial correlation (think regularity) as the ‘dimension’ increases. The code appears below, but is available for download in the class directory. The script will produce a surface plot of the field `B`, which is normalized so that its peak value is one, and is also then available to be used for simulating resource heterogeneity. The ‘power-law’ decrease in the spectral energy of waves, `p`, is defined as `p=(H+1)/2` because when the exponent is 1/2 the distribution is theoretically indistinguishable from ‘white’ noise. The most interesting structures appear for `H` between .5 and 1.5. An example of the kind of output you should expect is the cover art for this reader.

```
%
%       ROCKIES
%
%  This matlab script generates a structured but random spatial
%  environment of size NxN.  The user inputs the following:
%     N - number of grid points in both x,y directions
%     H - fractal dimension (bigger -> smoother the landscape)
%

  N=64; H=1.2;
  p=(H+1)/2;

  [I,J]=meshgrid([-N/2+1:N/2], [-N/2+1:N/2]); % wave modes
  randn('seed',0);
  phases=2*pi*rand(N);                   % random phases
  amp=randn(N).*(I.^2+J.^2).^(-p);       % normal  amplitudes about power law
  amp(N/2,N/2)=0;                        % avoid division by zero
  fA=amp.*exp(i*phases);                 % field in Fourier space

  A=ifft2(fA);                           % invert transform
  B=abs(A);                              % make field real, >0
  B=B/max(max(B));                       % normalize

  surf(B), colormap(jet), shading interp
  view(-37.5,60)
```

EXERCISE 20: Run the `rockies` program for a few choices of the ‘fractal dimension.’ Describe the relationship between increasing this parameter and the output field.

EXERCISE 21: Use the `B` field output by the `rockies` program to generate random landscapes of carrying capacity for the ENB model. One way to do this would be simply to execute the rockies script and then state

```
    k=2*B;
```

in the ENB program. Modify the graphic output to include contours of the resource with the density plots of the relative abundance of victims/parasites. What dynamics behaviors seem to correlate with what landscape features? What consequences do you think this might have for Nicholson's hypothesis for the persistence of parasitized populations in space?

Wow, Good Job! You now know secret and official things about the computational implementation of IDE using MATLAB *!*

4 Boundaries, Chemotaxis and Student Case Studies

4.1 Goals

In this lab we will finish off some of the complicated details which may be necessary for practical application of integro-difference equations (IDE). The first of these is the implementation of boundaries, in particular 'lethal' (or 'absorbing') boundaries and 'solid' (or 'reflecting') boundaries. The second is accounting for semi-random movement in response to external stimuli, in particular chemo-taxis. Thus, we will

- Learn how to use even and odd reflections to implement lethal ($P = 0$) and solid ($\frac{\partial P}{\partial x} = 0$) boundary conditions for IDE.
- Implement Powell et. al.'s 1998 approach for modelling chemotactic movement of a population.
- Spend time working out some of the computational details of the student case studies.

An overall goal for this class is to get students to 'work without a net,' that is, to create IDE simulations of ecological circumstances which may shed light on research questions. To encourage this, we will have lab time for teams to begin implementing an IDE approach, with our fearless instructor around to help with difficulties. Participants should work in teams of 2 or more to discuss the models which will be implemented, parameter regimes to be studied, and experimental protocols to be used. Then each student should build their own simulator and implement their portion of the experimental protocol, and get back together with the rest of the group to discuss results. I have suggested some group projects, but feel free to work on something closer to your own research or interests. Each group should have something to present to the rest of the class on the final day. You may have to work mainly with one spatial dimension in order to have enough time to experiment freely. As we have seen in the rest of the class, if you can do it in one dimension with MATLAB then you can do it in two dimensions with only slight modification.

4.2 Reflecting and Absorbing Boundary Conditions

4.2.1 The Wall of Doom

The general format of an IDE for a population P is

$$\begin{aligned} P_{t+1/2} &= f(P_t), \\ P_{t+1} &= K * P_{t+1/2}. \end{aligned}$$

Here f is the functional response connecting the dispersal stage organism (with density $P_{t+1/2}$) to the previous generation and K is the spatial dispersal probability for an initially localized individual. Spatial boundaries only have an impact in the second step during which dispersal occurs.

A lethal boundary condition is written mathematically as $P = 0$ at some $x = L$. This would seem to be quite a problem for our FFT-based approach, which assumes at a basic level that functions going into the FFT are periodic. However, we can get around this by *reflecting* the data. If the dispersal kernel, K, is symmetric (so that the probability of moving to the left is exactly the same as that of moving to the right, or $K(-x) = K(x)$), the effect of $P(x = L) = 0$ can be simulated by creating an *anti*-population of dispersers on the opposite side of $x = L$, which when it disperses back toward the positive (real) population will exactly cancel out at $x = L$. Since the functions we are looking at must be periodic on the entire (reflected) interval, we will also have a lethal boundary at $x = 0$ because of this reflection. Unlike what we have done before, the real population is only between $x = 0$ and $x = L$; the region from $x = -L$ to $x = 0$ only exists as a mechanism to implement the correct boundary conditions and has no physical interpetation. Below is a script which executes this implementation of boundary conditions.

```
%
%     BCs
%
%     matlab code to illustrate the effect of  boundary conditions
%     and their implementation
%
%     boundary conditions will be implemented  at x=0, x=xl
%     by reflecting the domain
%
%     p(x)   -    population density in x
%     K(x)   -    probability of individual dispersal to location
%                      x starting at location 0.
%     xl     -    maximum x coordinate (also - minimum)
%     dx     -    spatial step  delta x = 2*xl/np
%     np     -    number of points  in discretization for x
%                      (should be a power of 2 for  efficiency of FFT)
%     D      -    diffusion constant (per step)
%

    np=128; xl=5; dx=2*xl/np; c=0; D=.25;

%
%     discretize space, define an initial population distribution
%

   x=-xl + [ 0: np-1 ]*dx;
   p=( abs(x-1.5) <= 1. ); p0=p;  %  keep initial population
   plot(x, p0,'r')

%
%     define and normalize a dispersal kernel
%

   K=exp( -x.^2 /(4*D) )/sqrt( 4*pi*D );
   K=K/( dx*sum(K) );

%
%     calculate the fft of K, multiplying by dx to account
%     for the additional factor of np and converting from a
%     interval length of 1 to 2*xl.
%

   fK=fft(K)*dx;

%
%     skew-reflect the data to implement wall of doom
%       pa contains the actual population before dispersal
%       pr contains the reflected population before dispersal
%

   pa=p;
   pr=-p(np:-1:1);  %  start at np, go  backwards for reflection

%
%     calculate the fft of the pa, pr; perform the convolutions
%

   fpa=fft(pa);     fpr=fft(pr);
   fga=fK.*fpa;     fgr=fK.*fpr;

%
%     fgr and fga now contain FFT of the convolution of reflected
%     and actual fields, respectively; now invert
%

   gr=real( fftshift( ifft( fgr ) ) );
   ga=real( fftshift( ifft( fga ) ) );
   g=ga+gr;

%
%     update p, undoing the reflection (that is, take only the data
%     from x=0 to xl, with zeros elsewhere
%
   p=g.*(x>=0);

   hold on, plot(x, ga,'m', x, gr,'r', x, g, 'g', x, p, 'b'),  hold off
   axis([-xl xl -1 1])
```

Take particular notice of the statement

```
      pr=-p(np:-1:1);
```

which implements the reflection of the data. In MATLAB the statement `np:-1:1` will generate numbers starting at `np` and stepping backwards (by steps of -1) to 1. Placing these indices into `p`, as above, reflects the field (negatively) around $x = 0$. The program then disperses both the original and reflected fields, re-sums to get the field which implements the boundary conditions (`g` above), and then recovers the actual field (by setting the field back to zero for negative x). The results are plotted with the reflected field in red, the unreflected field in magenta, the sum of these two in green and the final result (the population after dispersal, with no negative individuals) in blue. Notice that neither the red (negative-reflected) nor magenta fields are zero at `0` or `xl`, but since they are skew-symmetric when they get added up they create the green field which *is* zero.

EXERCISE 22: Use `for ... end` statements and `pause` to continue the dispersal through several time steps. When you are confident that you have done this properly, extend the program to account for reproduction after dispersal (that is, replace `p=g.*(x>=0)` with `p=r*g.*(x>=0)`, where `r` is a parameter you will set to be larger than 1). What do you observe after iterating the combination of dispersal, lethal boundaries, and reproduction for several generations? What is the long term behavior when `r` is close to one? Can you adjust `r` so that the population persists? What is your interpretation?

4.2.2 Reflecting Boundaries

A reflecting boundary (that is, a boundary condition $\frac{\partial P}{\partial x} = 0$) is implemented the same way as a lethal boundary, except that the reflection is done *positively* instead of negatively. If the population projected on the far side of the wall is precisely the population on the physically relevant side, when the actual and reflected populations are summed the gradient must be zero at the wall itself. You can see this in the exercise below.

EXERCISE 23: To implement solid boundaries, in the script you have written above replace the line

```
pr=-p(np:-1:1);
```

with the line

```
pr=p(np:-1:1);
```

Run the script and see what happens, starting with `r=1`. If you iterate dispersal several times the final result should approach a constant, non-zero average value. Why is this? Remember that you have a solid wall on each end of the domain, which does not allow individuals or their offspring to escape. The effect of a long period of random motion, since no individuals are lost, is to eventually create a uniform distribution. Is there any problem with persistence in this case?

4.2.3 Mixed Boundaries

It might happen that one boundary is lethal ($P = 0$) while the other is solid ($P_x = 0$), in which case one needs to do a *double* reflection. Let's suppose we want to investigate a situation with a solid (reflecting) boundary at $x = 0$ and a lethal (absorbing) boundary at $x = L$. The absorbing boundary will require a skew reflection about $x = L$ (and a computational space twice as large as the physical space of interest), and this new field will be postively reflected about $x = 0$ to implement the reflecting boundary (resulting in a computational space four times as large as the original physical space). A MATLAB script implementing mixed boundary conditions (with drift and drop dispersal) appears below.

```
np=64;
xl=5;
dx=xl/np;
x=dx*[0:np-1];              % the physical space of interest
x2=dx*[-2*np:2*np-1];       % the virtual space on which reflections occur

D=.4;                       % per-step dispersal distance

K=.5/D*exp(-abs(x2)/D);     % double-exponential dispersal kernel
fK=dx*fft(K);

p0=(abs( x-2.5 ) <=1 );     % initial population

for istp=1:10,              % begin several dispersal steps

    pr=[p0, -p0(np:-1:1)];    % skew reflection to right for
                              %    lethal boundary at x=xl
```

```
        pl=[pr(2*np:-1:1), pr];   % positive  reflection to left for
                                  %    reflecting boundary at x=0

        fpl=fft(pl);
        p2=real( fftshift( ifft( fK.*fpl ) ) );

        pn=p2(2*np+1:3*np);       % use only that data corresponding to the
                                  %     physical region of interest

        hold  on, plot(x, p0,'r', x, pn,'g'), axis([0 xl 0 1]), hold off
        pause(.1)
        p0=pn;                    % update the population field

    end                       % of dispersal
```

Notice that implementing this kind of mixed boundaries requires vectors twice as long as implementing boundaries of the same type. This is not such a big problem in one dimensional simulations, but it can become computationally prohibitive in multiple dimensions.

4.2.4 Boundary Conditions in Two Dimensions

In principle, lethal and reflecting boundaries are implemented the same way in two dimensions as in one. Now, however, there are two sets of reflections to implement (in both x and y directions). Let's implement reflecting boundaries for a 'ring-random' or 'ripple' dispersal kernel,

$$K\left(r=\sqrt{x^2+y^2}\right)=Ce^{-\frac{(r-vt)^2}{4Dt}},$$

which models propagules leaving the place of their origin at speed v and random choice of direction, travelling and dispersing (with diffusivity D) for a time t before settling. This kernel (also called the 'ripple') was used by Brewster et. al. (1997) to investigate the dispersal of whiteflies in the Imperial Valley of California. The constant C must be evaluated numerically for normalization. The MATLAB script below implements 'ripple' dispersal with reflecting boundaries at $y=0$ and $y=L$ and lethal boundaries at $x=0$ and $x=L$. First, we set up both the grid of interest, `[X,Y]`, and the grid required for the reflections, `[X2,Y2]`.

```
%  set parameters for the ripple or ring-random diffusion kernel
%
   t=2; D=.1; v=2;
%
%  Define spatial parameters and 1-D coordinate axes
%
   xl=10; yl=10; np=64; dx=2*xl/np; dy=2*yl/np;
   x=linspace(0,xl-dx,np); x2=linspace(-xl,xl-dx,2*np);
   y=linspace(0,yl-dy,np); y2=linspace(-yl,yl-dy,2*np);
%
%  Define meshgrid for plotting physical space [X,Y]
%  and space in which reflections occur, [X2,Y2]
%
   [X,Y]=meshgrid(x,y);
   [X2,Y2]=meshgrid(x2,y2);
%
```

```
%  Define ring-random dispersal kernel and normalize
%
   K2D=exp(-(sqrt(X2.^2+Y2.^2)-v*t).^2./(4*D*t));
   K2D=K2D/(dx*dy*trapz(trapz(K2D)));           % normalize
%
%  Take FFT for purposes of convolution
%
   fK=fft2(K2D)*dx*dy;
```

This script sets up the grids and the ripple dispersal kernel. Now we can define an initial population (in a circle centered in the domain)

```
     p0=3*( ( (X-xl/2).^2 + (Y-yl/2).^2 ) <= 1. );
```

and perform reflections preparatory to doing the dispersal. First we must do an even reflection in the y direction for the reflecting boundaries:

```
     py=[p0; p0(np:-1:1,:) ];
```

The semi-colon above places one matrix above the other, and the `(np:-1:1,:)` takes rows (first index) in reverse order, for all columns (the colon in the second argument). This matrix must now be negatively reflected in the x direction to implement the lethal boundary

```
     px=[-py(:, np:-1:1) py];
```

Now we can just do the regular dispersal on the reflected grid:

```
   fp2=fft2(px);
   p2=real( fftshift( ifft2( fp2.*fK ) ) );
   pt=p2( np+1:2*np, np+1:2*np);
   pcolor(X,Y,pt), shading interp, colormap hot, axis square
```

You may want to try `surf(X,Y,pt), shading interp, colormap jet` to get a better idea how the boundaries are influencing dispersal The only particularly difficult thing here is knowing which part of the reflected and dispersed data to take to get the population of interest. Given the way that we did the reflection it is the ‘upper-right-corner’ of the data, or x and y indices from `np+1` to `2*np`.

EXERCISE 24: To make sure that you understand the reflection process, edit the script above so that it implements a lethal boundary at $y = 0, L$ and a reflecting boundary at $x = 0$ and $x = L$. How many reflections would you need to implement a reflecting boundary at *only* $x = 0$, with lethal boundaries on the other three?

EXERCISE 25: Write a script which will do several iterations of ring-random dispersal with short time steps and plot the results to see what happens as the waves of dispersal approach the boundaries.

4.3 A Model for Chemo-Taxis

The last thing that might be of interest is implementing chemotaxis. Many (if not most) insect species find mates, hosts, or prey (or all) using chemical clues. Bark beetles find hosts and initialize a ‘mass-attack’ using a pheromone feedback system (Powell et. al. 1998), natural populations of *Drosophila* find rotting fruit to lay eggs on by following the odor of ethanol (Etienne et. al 2002, Lof et. al 2008), and a variety of insects, including ladybird beetles (*Coccinella*) respond to the odor of sugar-water. Powell and co-workers (1998) developed an IDE approach to emulate the chemotactic dispersal process, which is discussed here in one dimension in the context of *Drosophila* response to ethanol.

Consider a resource (apples) distributed with density R, fermenting and releasing ethanol (E) at a rate δ. The steady-state ethanol distribution then satisfies

$$E = K_E \star (\delta R)\,, \quad K_E = \frac{1}{2L}\exp\left[-\frac{|x|}{L}\right],$$

where L is the mean dispersal distance of the ethanol. A model for the 'sensory index', or degree of saturation of the sensory apparatus of the flies, is

$$F(E) = \frac{E}{E_0 + E}$$

Here E_0 is a saturation parameter – basically the level at which the sensory apparatus of the *Drosohpila* is halfway to being completely saturated with ethanol. A model for the population response is

$$\frac{\partial}{\partial t}P = -\frac{\partial}{\partial x}\left[\overbrace{\nu P\frac{\partial F(E)}{\partial x}}^{\text{toward increasing } F} - \overbrace{\mu\frac{\partial P}{\partial x}}^{\text{random motion}}\right].$$

The parameters ν and μ parameterize the relative strengths of the tactic response and random dispersal, respectively. Let

$$K_R = \frac{1}{\sqrt{4\pi\mu t}}\exp\left[-\frac{x^2}{4\mu t}\right]$$

be the random dispersal kernel for flies (when $\nu = 0$ or $E = 0$), implemented over some time interval, t. An approximate solution (details of how approximate are discussed in Powell et. al., 1998) is given by

$$\begin{aligned}\Pi(x,t) &= K_R \star \left\{\exp\left[-\frac{\nu}{\mu}F(E)\right]P(x,t=0)\right\},\\ P(x,t) &= C\exp\left[\frac{\nu}{\mu}F(E)\right]\Pi(x,t).\end{aligned}$$

The constant C must be chosen to normalize $P(x,t)$ because this method is not guaranteed to preserve the number of individuals in the dispersing population. Following is a MATLAB code which implements the chemotactic procedure over several iterations.

```
%
%       CHEMOTAX
%
%       matlab code to emulate a chemotactic process
%
%       p(x)    -       population density in x (drosophila)
%       e(x)    -       ethanol concentration
%       r(x)    -       resource distribution
%       KR(x)   -       probability of individual dispersal
%       KE(x)   -       dispersal kernel of ethanol
%       xl -        maximum x coordinate (also - minimum)
%       dx -        spatial step  delta x = 2*xl/np
%       np -       number of points  in discretization for x
%       mu -       diffusion constant (per step)
%       nu      -       chemotactic constant
%       a       -       dispersal distance of ethanol
%       d       -       release rate of ethanol from resource
%       E0      -       saturation constant for sensory index
%
   np=128; xl=8; dx=2*xl/np;
   t=1; mu=1; nu=10;
   a=.5; d=2; E0=.5;
   nsteps=15;
%
%       discretize space, define an initial population distribution
%
   x=-xl+[0:np-1]*dx;
   p=(abs(x+3)<=.5);   p0=p;   %  keep track of initial p for comparison
   r=(abs(x-2)<=.5);           % resource density
%
%       define dispersal kernels and normalize
%
   KR=exp(-(x).^2/(4*D*t))/sqrt(4*pi*D*t);
   KE=exp(-abs(x)/a);
   KR=KR/(dx*sum(KR));
   KE=KE/(dx*sum(KE));
%
%       calculate the fft of KR, KE; dx normalizes convolution
%
   fKR=fft(KR)*dx;
   fKE=fft(KE)*dx;

%       calculate ethanol field

   E=real( fftshift( ifft( fft( d*r ).*fKE) ) );
   plot(x, p,'b', x, p,'m--', x, E/max(E),'y', x, r,'r'), ...
        axis([-xl xl 0 1.1*max(p)])

%       calculate sensory index

   FE=E./(E0+E);

%       begin iterating the model

   for j=1:nsteps,
%
%       calculate the fft of pi = p.*exp(-nu/mu f(E));
%       perform the convolution on the pi
%
      peye=p.*exp(-nu/mu*FE);
      fpi=fft(peye);
      fg=fKR.*fpi;
%
```

```
%        fg now contains the fourier transform of the convolution;
%        invert it, multiply by the inverse exp(nu/mu*FE)
%
      g=exp(nu/mu*FE).*real(fftshift(ifft(fg)));
%
%        Now we must be careful to normalize!
%
      C=trapz(p)/trapz(g); g=C*g;

      hold on, plot(x,g,'b'), hold off
      pause(.1)

%        update p and move to the next time step
      p=g;
   end               % of the iterations
```

One can think of this implementation as a pre-multiplication of the dispersing population, P, with a dispersal 'bias': individuals further away from the source of the ethanol (therefore sensing low $f(E)$) are more strongly biased to random dispersal. The biased population (Π) is dispersed at random, and then de-biased (by multiplication of $\exp[\nu/\mu f]$). Provided the time steps are small enough, this approximates the chemotactic dispersal process. As it turns out, the model works just fine for large time steps, even if it does not approximate the PDE. As with the prey-taxis discussed in the previous lab (Turchin-taxis) the main difficulty is that the process is not linear in t– five steps with $t = 2$ is not necessarily equivalent to two steps with $t = 5$.

EXERCISE 26: Form hypotheses about the impact of the parameters on chemotactic dispersal, and test your hypotheses against model iterations. For example, for a given time step, if the ethanol disperses further (larger `a`) the flies should find the resources more rapidly. Or, if chemotaxis is stronger than random motion ($\nu > \mu$) then the resulting pattern of aggregation should be much more tightly centered on the resource. Change parameters to reflect your hypotheses and test if they are correct.

EXERCISE 27: Adapt the chemotactic procedure to work in two dimensions (really the only thing that needs to change is the definition of space and working out the 2D FFT). Try using the `rockies.m` program to give a random background environment for ethanol production.

Well, look at you! You have gotten to the end of the formal lab material on integro-difference equations! Good Work!

References

[1] Allen, J.C., C.C. Brewster and D.H. Slone, 2001. Spatially explicit ecological models: a spatial convolution approach. *Chaos, Solitons and Fractals* 12: 333–347.

[2] Allen, J.C., C.C. Brewster, J.F. Paris, D.G. Riley and C.G. Summers, 1996. Spatiotemporal modeling of whitefly dynamics in a regional cropping system using satellite data. Pages 111–124 in: Gerling, D. and R.T. Mayer, eds. *Bemisia 1995: Taxonomy, Biology, Damage, Control and Management.* UK: Intercept, Andover.

[3] Allen, L.J.S, E.J. Allen and D.N. Atkinson, 1999. Integro difference equations applied to plant dispersal, competition and control, *Fields Institute Communications* 21: 15–30.

[4] Andersen, M., 1991. Properties of some density-dependent inegrodifference equation population models. *Mathematical Biosciences* 104: 135–157.

[5] Andow, D.A., P.M. Karieva, S.A. Levin and A. Okubo, 1990. Spread of invading organisms. *Landscape Ecology* 4: 177-188.

[6] Brewster, C.C. and J.C. Allen, 1997. Spatiotemporal model for studying insect dynamics in large-scale cropping systems. *Environmental Entomology* 26: 473–482.

[7] Cain, M.L., H. Damman and A. Muir, 1998. Seed dispersal and the Holocene migration of woodland herbs. *Ecological Monographs* **68**: 325–347.

[8] Clark, J.S., M. Silman, R. Kern, E. Macklin and J. HilleRisLambers, 1999. Seed dispersal near and far: patterns across temperate and tropical and tropical forests. *Ecology* **80**: 1475–1494.

[9] Clark, J.S., C. Fastie, G. Hurtt, S.T. Jackson, C. Johnson, G.A. King, M. Lewis, J. Lynch, S. Pacala, C. Prentice, E.W. Schupp, T. Webb III and P. Wyckoff, 1998. Reid's paradox of rapid plant migration. *Bioscience* **48**: 13–24.

[10] Cobbold, C.A., Lewis, M.A., Roland, J., Lutscher, F. 2005. How parasitism affects critical patch size in a host-parasitoid system: Application to Forest Tent Caterpillar. *Theoretical Population Biology* 67: 109-125.

[11] Edelstein-Keshet, L. 1988. *Mathematical Models in Biology*. McGraw-Hill, New York, 586 pages.

[12] Etienne, R., B. Wertheim, L. Hemerik, P. Schneider and J. Powell, 2000. "Dispersal may enable persistence of fruit flies suffering from the Allee effect and scramble competition," *Proceedings of the Dutch Entomological Society* (11): 121–128.

[13] Etienne, R., Wertheim, B., Hemerik, L., Schneider, P., Powell, J.A., 2002. The interaction between dispersal, the Allee effect and scramble competition affects population dynamics. Ecol. Model. 148(2), 153168.

[14] Hart, D.R. and R.H. Gardner, 1997. A spatial model for the spread of invading organisms subject to competition. *J. Math. Biology* 35: 935–948.

[15] Holmes, E.E., M.A. Lewis, J.E. Banks and R.R. Veit, "Partial differential equations in ecology: spatial interactions and population dynamics," *Ecology* 75: 17-29.

[16] Heavilin, J., J.A. Powell and J.A. Logan. 2007. "Development and parametrization of a model for bark beetle disturbance in lodgepole forest." pages 527-553 IN: K. Miyanishi and E. Johnson (eds), *Plant Disturbance Ecology*, Academic Press, NY.

[17] Heavilin, J. and J.A. Powell, 2008. A novel method of fitting spatio-temporal models to data, with applications to the dynamics of Mountain Pine Beetles. Natural Resource Modeling 21: 489-524.

[18] Kot, M., 1992. "Discrete-time travelling waves: Ecological examples," *Journal of Mathematical Biology* 30: 413–436.

[19] Kot, M.A. Lewis and P. v.d. Driessche, 1996. Dispersal data and the spread of invading organisms. *Ecology* 77: 2027–2042.

[20] Lewis, M.A. and J.D. Murray, 1993. Modelling territoriality and wolf-deer interactions. *Nature* 366: 738–740.

[21] Lof, M.E., Etienne, R., de Gee, M., Hemerik, L., Powell, J., 2008. The effect of chemical information on the spatial distribution of fruit flies: I Model results. *Bull. Math. Biol* 70:1827-1849.

[22] Luckinbill, L.S., 1973. Coexistence in laboratory populations of *Paramecium aurelia* and its predator *Didinium nasutum*. *Ecology* 54: 1320–1327.

[23] Murray, J.D., 1989. *Mathematical Biology*. Springer-Verlag, Berlin, 767 pages.

[24] Neubert, M.G., M. Kot and M.A. Lewis, 1995. Dispersal and pattern formation in a discrete-time predator-prey model. *Theoretical Population Biology* 48: 7–43.

[25] Nicholson, A.J., 1957. The self-adustment of of populations to change. *Cold Spring Harbor Symp. Quantum Biol.* 22: 153–172.

[26] Pielaat98 Pielaat, A. and F. van den Bosch, 1998. A model for dispersal of plant pathogens by rain-splash. *IMA J. Mathematics Applied in Medicine and Biology* 15): 117–134.

[27] J.A. Powell, I. Slapničar and W. van der Werf. 2005. “Epidemic Spread of a Lesion-Forming Plant Pathogen – Analysis of a Mechanistic Model with Infinite Age Structure,” *Journal of Linear Algebra and Applications* 398: 117-140.

[28] Powell, J.A. and N.E. Zimmermann, 2004. “Multi-Scale Analysis of Active Seed Dispersal Contributes to Resolving Reid’s Paradox,” *Ecology* 85:490-506.

[29] Powell, J., T. McMillen and P. White, 1998. Connecting a Chemotactic Model for Mass Attack to a Rapid Integro-Difference Emulation Strategy, *SIAM J. Appl. Math* vol. 59, no. 2, pp 547-572.

[30] P. Skelsey, W.A.H. Rossing, G.J.T. Kessel, J. Powell, and W. van der Werf. 2005. “Influence of Host Diversity on Development of Epidemics: An Evaluation and Elaboration of Mixture Theory,” *Phytopathology* 95:328-338.

[31] Tilman, D., 1994. Competition and biodiversity in spatially structured habitats. *Ecology* 75: 2-16.

[32] Turchin, P., J.D. Reeve, J.T. Cronin and R.T. Wilkens, 1997. Spatial pattern formation in ecological systems: bridging tehoretical and empirical approaches. Pp. 195-210 *in:* Bacompte J., Solé R.V. (eds), *Modelling Spatiotemporal Dynamics in Ecology*. Landes Bioscience, Austin, TX.

[33] Turchin, P., 1989. Population consequences of aggregative movement. *J. Animal Ecology* 58: 75–100.

[34] van den Bosch, F., J.A.J. Metz and J.C. Zadoks, 1997. “Pandemics of focal plant disease, a model,” *Technical Note 97-06*, *Wageningen Agricultural University*, 22 pages.

[35] van der Werf, W., E.W. Evans and J. Powell, 2001. Measuring and modelling dispersal of *Coccinella septempunctata* in alfalfa fields. *European Journal of Entomology* 97: 487–493.