EZRide
A Code for Simulating Spiral Waves in Comoving Frame of Reference

Andrew J. Foulkes & Vadim N. Biktashev

March 3, 2010

1 Introduction

EZRide is a program which simulates spiral waves in a frame of reference which is moving with the tip of the wave. The motivation behind the program is within [3]. The implementation is based on the very popular program EZ-SPIRAL by Dwight Barkley [2, 1]. The present public issue of the code is intended as supplementary material for our paper "Riding a spiral wave: Numerical Solutions of Spiral Waves in a Moving Frame of Reference" submitted to Physical Review E in January 2010. We also refer to [4] for some prior mathematical background of solving spiral waves in the comoving frame of reference.

EZRide uses both Barkley's and FitzHugh-Nagumo's (FHN) models, together with a choice of boundary conditions (Neumann or Dirichlet) and whether interactive graphics are to be used or not.

The following set of intructions are organised as follows:

Section 2 Getting Started: This will talk you through how to get the program working by following two examples.

Section 3 User's Manual: This will decribe the important uses of EZRide and how the user can do particular tasks.

Section 4 Programmer's Manual: This section will describe how the program works, detailing the differences between EZRide and EZ-SPIRAL, the function of each file, and descriptions of the main functions used.

2 Getting Started: A Quick Guide

We will talk you through getting the program started by following a couple of examples - one example for a rigidly rotating spiral wave and another example for a meandering spiral wave. We will initial choose the parameters such that fast numerical calculations are illustrated. However, we will see that the tip trajectories recontructed from the quotient data ("quotient data" is the name given to the coefficients to the advection terms — see [3] for details) are not very accurate. So, we provide details at the end of this section of changes to be made to the numerical parameters in order to get a more accurate reconstruction of the

tip trajectory (note that the refined parameters still provide for fast numerical calculations, but obviously not as fast as first illustrated).

At each step, issue the command to the left of the table (you will obviously need to have a terminal screen open and you will also need to be in the directory which contains the program files). The commentary to the right of the table describes what each command does.

Before we start, you will have to adjust `Makefile` to reflect the environment you are working in. We have developed the code in Mac OSX. We hope with suitable modifications it should work in any unix-type operating system with X11 graphics. For instance, the compiler used is shown by the flag `CC` on line 42, which is initially set to `CC=cc`. Adjust this as you need to.

We are now ready to work through the examples. At each line in the tables below, issue the command shown to the left.

Command	Commentary
`make ezride`	This makes the program `ezride` using Barkley kinetics and interactive graphics (these are the default settings in the `Makefile` - see Sec.(3.4) for further details).
`./ezride`	Executes the program `ezride`. An X-Window should appear.
Select the X-Window	In order to run the simulation, the X-Window displaying the simulation needs to be the Window on the system which is on top of all others, and get the keyboard focus. This can be achieved by clicking on the window with the mouse, placing the mouse arrow within the window or tabbing until the window comes to the top.
Press `SPACE` bar	This will initiate the simulation.
Press `t` key	This will display the tip path from time of pressing the key. At the beginning of the simulation, you may notice several tips. Press `t` twice to erase the tips and start the drawing process again. You should notice a smooth tip pattern being plotted.
Press `r` key	Switches on the **riding mode**. That means, the advection terms will be added to the right hand side, to describe evolution in a moving frame of reference. Do this once the spiral wave has fully formed, say when the tip has traced out one full circle.

Now, leave the simulation to run until it self-terminates (initially, the number of time steps have been set at 50,000 and so the program will self terminate when the number of time steps has been exhausted). You should notice that the spiral wave within the comoving frame becomes stationary. This means that we have a Rigidly Rotating Spiral Wave.

The simulation, can be brought to an end by several means: pressing keys `q`, `esc`, exiting the X-Window by physically closing it, or when the number of time steps has been exhausted. The current directory will then contain, depending on the task settings, apart from the usual EZ-SPIRAL files (`fc.dat`, `tip.dat`) also a file called `quot.dat`, containing the quotient data, i.e. c_x, c_y and ω as functions of time, and the file called `recon.dat`, which contains reconstructed trajectory of the tip (see detailed description below).

We will now look at a meandering wave using FHN. We note that there are two different methods of recompiling the program for different kinetics and these methods are described in Sec.(3.4).

Command	Commentary
`make ezride KINETICS='FHN'`	This makes the program `ezride` using FHN kinetics and interactive graphics.
`./ezride`	Executes the program `ezride`.
Bring X-Window to top	Needs to be done in order to observe the results of simulation, and also to be able to pause or prematurely stop it, and to do a variety of other tasks.
No need to press `SPACE` bar	Starts automatically.
No need to press `t` key	Displays the tip path automatically

This time, let the simulation self terminate, without having advection switched on. We will now generate initial conditions by using the final conditions from the last run (saved in `fc.dat`). Also, we will copy the tip file to another file so that the data generated (which we will use to plot the original tip trajectory), is not overwritten.

Command	Commentary
`cp fc.dat ic.dat`	Copies the final conditions of the previous simulation to the initial conditions for future simulations.
`cp tip.dat tip_fhn.dat`	Copies the tip data file to another data file.
Edit file `task_fhn.dat`	Change line '`0,20,0  initial_ride and` ...' to '`1,20,0  initial_ride and` ...'
`./ezride`	Executes the program `ezride`.

The program will start automatically in the riding mode. Allow it to self terminate. Now you can compare the reconstructed tip trajectory found in the `recon.dat` when the one you have saved in `tip_fhn.dat`.

You should get two flower patterns produced which are qualitatively similar, as in the paper.

We noted at the beginning of this section that we have chosen parameters which will illustrate just how fast this program can be. However, the accuracy at such crude resolution can be low, and numerically accurate results require finer resolution, as discussed in the paper.

3 How it works - The User's Manual

This section will detail how to perform certain tasks using EZRide. We will not go into the detail on how the code actually works (this is covered in the next section) but just simply how to perform a task.

3.1 Task Files

There are two task files which hold the important parameters, viz. `task_fhn.dat` for FHN kinetics and `task_bark.dat` for Barkley kinetics. The structure of the files are similar to `task.dat` in EZ-SPIRAL and the files themselves are briefly commented. Here we provide

some further comments on the parameters that are different from EZ-SPIRAL. Some of the difference are essential for the function of EZRide as opposed to that of EZ-SPIRAL, and some are merely for convenience.

- Line 6: Boundary conditions as set initially. Note this can be switch in the interactive mode.
- Line 12: Two comma-separated integer numbers, which define the displacement of the position of the cross-field initial condition from the centre of the box. This can be used to arrange the spiral wave in the desirable position within the box in a non-interactive way.
- Line 14: Three comma-separated integer numbers. If the first is nonzero, the calculations start in riding mode; this can be switched in the interactive mode. The other two numbers are grid coordinates of the second pinning point with respect to the box center. Note that the first pinning point is always fixed to the box centre.
- Line 16: An integer and two floating-point numbers, comma separated. If the integer is nonzero, the perturbation is on (this can be switched on/off in the interactive mode), and the two floats are its amplitudes, $E_{1,2}$, by components. The perturbation is an extra term of the form $+A_i \partial u_i/\partial x$ in each of the two right-hand sides of the reaction-diffusion equations, $i = 1, 2$.
- Line 20: If nonzero, then image files are generated in PNG format, named sequentially `000000.png`, `000001.png`, in every so many time steps. Note that images can also be generated in response to dialogue commands.
- Line 21: the usual EZ-SPIRAL history flag and history point coordinates are combined here on one line, comma-separated.
- Line 23: if nonzero, the quotient data are written to file `quot.dat` as well as to the standard output.
- Line 25: `initial_field` : in addition to values 0 and 1 as in EZ-SPIRAL, we use value 2 to show "phase" view: red colour component to represent the "u-field", and the green and blue colour components to represent the v-field.
- Line 26: Here the tip detection flag can be switch on in advance, not only interactively.
- Line 29: `autostart` : if nonzero, the program starts running automatically without waiting for the user to press the `SPACE` button.
- If `initial_field`$= -1$ and `autostart`$\neq 0$, then the program will run without graphics and in non-interactive mode, even if it was compiled with `GRAPHICS=1` in the Makefile.

For the remainder of this report, we will shorten the names of the task files to `task_kin.dat`.

3.2 The interactive (dialogue) mode

Once the program has been started and the graphics is on, its behaviour can be controlled by pressing keys on the keyboard. Note that the present code can be used *without* dialogue and graphics, in a batch mode, by setting appropriate flags in the task file (see below).

In some systems, it is necessary to have the mouse arrow within the X-Window for the key press to take effect, and/or to click this window with the mouse. The key presses are detailed in the following table (NB uppercase letters work in the same way as the corresponding lowercase letters):

key	action
`SPACE`	start/continue the simulation
`p`	pause the simulation (then `SPACE` will restart it)
`q`	quit the simulation and save the final conditions data
`esc`	quit the simulation but not save the final conditions data
`u`	show the u field
`v`	show the v field
`f`	show the phase field, a combination of the u and v fields as described above
`n`	show no field (useful when you only want to see the tip trajectory)
$\uparrow$, $\downarrow$, $\rightarrow$, $\leftarrow$	shift the spiral in a particular direction
`t`	toggle tip on/off: show the tip path from time of key press
	(when in riding mode, show pinning points instead of tip path)
`b`	toggle between Neumann and Dirichlet boundary conditions
`h`	toggle perturbation on/off
`r`	toggle riding mode on/off
`s`	save the copy of the X-Window to a `png` file
`i`	toggle autmatic saving X-Window images.
`m`	generate a `png` file regardless of XWindow system

3.3 Output Files

3.3.1 File tip.dat

This file contains all the data relating to the tip position and orientation. There are four columns within the file relating to (from left to right) time t, coordinates X and Y of the tip, and tip orientation angle Θ. This is no different from EZ-SPIRAL, except for the fourth column which was not there, and that when in riding mode, the tip position is fixed at or very close to the pinning point.

3.3.2 File quot.dat

This file contains the values of the advection coefficients (the quotient data) c_x, c_y and ω. It is organised as four columns as (from left to right) time t, c_x, c_y and ω.

3.3.3 File recon.dat

The meaning of the 11 columns of this file is:

- The time, t,

- The tip position with respect to the current Frame of Reference (FoR), X_0, Y_0, Θ_0.
- The current FoR position with respect to the laboratory FoR, X_1, Y_1, Θ_1.
- The resulting tip position with respect to the laboratory FoR, X_2, Y_2, Θ_2. The triple (X_2, Y_2, Θ_2) defines a Euclidean transformation which is a superposition of the two Euclidean transformations defined by triples (X_0, Y_0, Θ_0) and (X_1, Y_1, Θ_1).
- The flag indicating whether the calculations are in the "riding mode" (1) or not (0).

3.3.4 File history.dat

This is used to track the values of u and v at a particular grid point and contains three columns time, u and v. It is no different from EZ-SPIRAL.

3.3.5 File fc.dat

The structure of the file is mostly similar to that in EZ-SPIRAL. The first two lines contain the model parameters used in that simulation, and also the numerical parameters. The third line is different from EZ-SPIRAL: it saves the "riding coordinates", that is the final position (X_1, Y_1, Θ_1) of the working FoR with respect to the laboratory FoR, which is used for continuity of the tip reconstruction in case this file is used as an initial condition for subsequent runs. The fourth line gives the date and time the file was formed. Lines 5-9 are reserved for further comments. The first character of line 10 determines whether the data are stored as ASCII or binary, and lines 11 onward contain the u and v values of each grid point starting from bottom left and working right and upwards.

3.4 Change Kinetics & switch graphics on or off

Changing both the kinetics and whether the program is run in the graphics mode is done via the `Makefile`, so no need to amend the code. There are two different ways to do this according to your preference:

3.4.1 Method 1: Command Line Compilation

The Makefile contains two important macros which indicate which kinetics to use (`KINETICS`) or whether graphics are to be used (`GRAPHICS`). `KINETICS` takes the values `Barkley` or `FHN`, which obviously compile the program using Barkley's or FHN's kinetics respectively. Whilst `GRAPHICS` takes the values 1 for interactive graphics, or 0 for no graphics.

So, we can take advantage of overriding the values of the macros within the `Makefile` via the command line as shown in the following table

command	program compiled with:
`make ezride KINETICS='Barkley' GRAPHICS='1'`	Barkley and Interactive Graphics
`make ezride KINETICS='Barkley' GRAPHICS='0'`	Barkley and No Graphics
`make ezride KINETICS='FHN' GRAPHICS='1'`	FHN and Interactive Graphics
`make ezride KINETICS='FHN' GRAPHICS='0'`	FHN and No Graphics

If the command `make ezride` is issued, then the program is compiled with the value of the macros set within the `Makefile`. So, by default, the program comes with the macros set at `KINETICS=Barkley` and `GRAPHICS=1`, i.e. Barkley kinetics and interactive graphics.

Also, if the user is not confident to amend the `Makefile` to change the C compiler macros (`CC`) from the default `CC=cc` to their systems own compiler, for example to `gcc`, then this can be done as:

```
make ezride KINETICS='Barkley' GRAPHICS='1' CC=gcc
```

NOTE: To run EZRide with no interactive graphics, you may prefer to already have a set of initial conditions. Therefore, it is recommended that you run the program with graphics, switch on advection, let it settle down (after an initial transient period) and copy the final conditions file to initial conditions (`cp fc.dat ic.dat`).

3.4.2 Method 2: Amending the Values of the Macros within the Makefile

So, the second way is to actually change the values of the `Makefile` macros mentioned in method 1 to the values the user requires.

Open the `Makefile` using your favourite text editor. To compile the program with Barkley kinetics, line 25 should be uncommented (i.e. no '#' should be at the beginning of the line), and make sure that line 26 is commented out (i.e. there is a '#' at the start of the line). Save and close the file and remake the program by issuing the command `make ezride`.

To change the graphics, you will need to change the value of `GRAPHICS` in line 19 to 1 for interactive graphics or 0 for no graphics. Save and close the file and then issue the command `make ezride`.

3.5 Initial Conditions

To generate initial conditions from scratch, firstly remove any current initial conditions file `ic.dat` (or rename it, if it is valuable). Start EZRide by issuing the command `./ezride`, and pressing `SPACE` if `autostart` is not set on in the task file. You should see your spiral forming, obviously depending on your choice of parameters.

Ideally, you should try and get your spiral tip rotating around the center of the box (keeping it away from the boundaries). Therefore, you can adjust the initial position of the cross stimulation fields by one of two methods. You can firstly issue `./ezride` and then use the arrow keys to move the fields, such that the wave ends up, once started, rotating about a point close to the center, or you can use the parameters in the files `task_kin.dat` in line 14 to move the field prior to the program being executed.

The next way to use initial conditions is to use the final conditions from the previous run as initial conditions for the next. This is done by issuing the command `cp fc.dat ic.dat`.

3.6 Change Box Size

The box size is amended in `task_kin.dat`. The parameter is called `Lx` and is located on line 4. Note that this code only operates with square grids.

Also, to perserve a particular spacestep, you will need to change `nx` in line 8 accordingly.

3.7 Change Timestep

The timestep can be changed by amending `ts` in the `task_kin.dat` file on line 8. Parameter `ts` is time step as fraction of diffusion stability limit so, for five-point Laplacian, $\Delta_t = \texttt{ts} \times \Delta_x^2/4$.

3.8 History File

If you need to observe a particular feature of the spiral wave by monitoring how u and v change at a selected "history point" over time, then you can activate the history file.

Open up `task_kin.dat`. On line 21, you can specify how many time steps per write to the history file (called `history.dat`). If this is zero then no history file is written. Further on the same line, specify the grid coordinates of the history point, measured from bottom left corner.

3.9 Changing Boundary Conditions

In the interactive mode, this is done via pressing `b`. The default BC's are set via the `task_kin.dat` file on line 6: 0 for Dirichlet, 1 for Neumann.

3.10 Change Position of Second Pinning Point

The first number on line 14 determines whether the riding mode is on when the program starts. The second and the third numbers on the same line are of parameters called `i_inc` and `j_inc`. They define the position of the second pinning point with respect to the first pinning point, in grid coordinates. The first pinning point is always assumed in the centre of the box. Note that the distance between the pinning points affects the accuracy of pinning (the further, the more accurate) and stability (may become unstable if too far), while the direction of the vector `(i_inc,j_inc)` determines orientation of the spiral with respect to the box when in the riding mode.

WARNING: Setting `i_inc` and `j_inc` to zero simultaneously will give a division by zero if advection is switched on (i.e. by pressing `r`). When the program is executed, a warning is displaying on the terminal screen if they are zero. The program will still run while in the non-riding mode (stationary FoR), but will terminate once advection is switched on.

4 How it works - The Programmer's Manual

This section will describe how particular parts of the code work. We firstly detail the main differences between EZRide and EZ-SPIRAL.

4.1 Differences between EZRide and EZ-SPIRAL

1. Riding mode — this is the most important difference. Advection terms are calculated via the function `step1()` found in `ezadvec.c`.

2. Perturbation — for this public issue of the code, we include one sort of perturbation, "electrophoresis". The corresponding function `Perturb()` is found in `ezdrift.c`. Other sorts of perturbations can be incorporated similarly using the formulas presented in the paper.

3. Kinetics — FHN kinetics have been incorporated into EZRide via `ezstep.h`. Also, different sets of parameters are needed for FHN compared to Barkley kinetics and therefore this is incorporated throughout the code.

4. Boundary Conditions — Dirichlet BC's have been incorporated. The boundary conditions are specified in the task file and switched by pressing `b`.

5. Phase Field — this is visualization using a combination of the u and v-fields, which is selected by setting `initial_field` to 2 in the task file or by pressing `f`.

6. Output — `tip.dat` now holds the orientation of the tip as well as its spatial coordinates; new output file `quot.dat`, created when advection is calculated, holds the values of the coefficients of the advection terms, i.e. c_x, c_y and ω; new output file `recon.dat` presents reconstructed tip trajectory.

7. Saving Images — the user can now save just a single copy of the window to a `png` file (pressing `s`) or a series of images to a series of `png` files (pressing `i`). This should work in most unixes with X11 graphics. In case it does not, and for work in non-graphical mode, we also have a procedure of image generation procedure in `ppm` format with a fork to a `netpbm` pipeline converting it to `png`.

8. Task Files — there are now two task files, one for Barkley kinetics, one for FHN. The program will choose which is necessary at the compile time, depending on which kinetics are used.

9. `Makefile` — has now options to make the program with a particular set of kinetics and whether interactive graphics are used.

4.2 Files

File	Description
`ezride.c`	Overall control, files i/o, memory allocation.
`ezstep.c`	Most timestepping and boundary conditions.
`ezadvec.c`	The substep representing the advection (FoR movement).
`ezdrift.c`	The substep representing the perturbation.
`ezgraphGL.c`	All the graphics.
`eztip.c`	Finding the tip position and orientation.
`ezride.h`	All the global variables and some of the major macros.
`ezstep.h`	Mainly the algorithms for the kinetics.
`ezgraphGL.h`	Macros for the graphics.
`Makefile`	Contains instructions how to compile the code etc.
`task_bark.dat`	Sample task file for Barkley kinetics.
`task_fhn.dat`	Sample task file for FHN kinetics.
`ezride.pdf`	This document.

It is intended that the user will not only use the program as it is, but will amend it to suit their needs. We therefore detail in the following subsections, the important functions that are used in EZRide, including where they are found and what they do. We focus on the differences compared to EZ-SPIRAL.

4.2.1 Makefile

We assume that the user is sufficiently fluent in UNIX commands so that they can amend the Makefile to suit their needs and their operating environment, in particular the lines in `Makefile` which refer to their systems, e.g. the C compiler, location of the libraries etc.

The two major macros integrated within `Makefile` are `KINETICS` and `GRAPHICS`, which determine which Kinetics are used and whether to use graphics respectively. The values of these macros are passed to the compilation command line via the macro `CFLAGS` using the preprocessor command `-DFHN=$(KIN)` and `-DGRAPHICS=$(GRAPH)`, where `KIN` is either 1 (for FHN kinetics) or 0 (for Barkley), and `GRAPH` is either 1 (for graphics) or 0 (for no graphics).

4.2.2 ezride.c

This file contains the `main()` function. Within the main function the whole program is initialised (via `Initialise()`), and then the main loop is executed. The loop is a bit more complicated than in EZ-SPIRAL because of three substep in the timestepping, and more of i/o calls.

Also in the `main()` function there is a timer which calculates the time the whole simulation has taken (in real time). This will prove useful when comparing the time taken for particular simulations.

Functions `cube()` and `root()`, calculate the steady state in FHN, which are then used in the drawing procedure.

Function `Write_quot()` writes the quotient data (c_x, c_y and ω) to a file called `quot.dat`.

Function `Recon_tip()` calculates and records the reconstructed tip position to a file called `quot.dat`, taking into account the accumulated displacement of the working FoR with respect to the laboratory FoR.

The way that initial conditions are generated via `Generate_ic()` has been amended slightly so that the position of the cross field stimulations can be controlled via the variables found in `task_kin.dat` on line 12.

Finally, function `Make_images` generates png images independently of X11.

4.2.3 ezride.h

This file is the main header file, used in all the other files, and contains mainly macros used throughout the program and also declares the global variables.

4.2.4 ezstep.c

There are two main functions contained in this file, with the same purpose as originally contained in EZ-SPIRAL.

The first is `step()`. It's purpose is to calculate the values of $u(x, y, t)$ and $v(x, y, t)$ for each time step for the Reaction-Diffusion part of the equation. It has been modified to allow FHN kinetics.

The second is `Impose_boundary_conditions()` which implements boundary conditions. It has been modified to allow Dirichlet boundary conditions.

4.2.5 ezstep.h

This relatively short file contains all the macros for the kinetics used in `step.c`.

4.2.6 ezadvec.c

This provides the function `step1()`, which implements the advection substep, describing the FoR movement, when the riding mode is on.

4.2.7 ezdrift.c

This provides the function is `Perturb()`, which implements the perturbation substep, when the perturbation mode is on.

4.2.8 eztip.c

This file is entirely devoted to finding the tip position. The changes compared to EZ-SPIRAL are: the orientation of the tip is calculated, and in case of multiple tips, all the tips, except the last found, are discarded. Note that the present code is intended for use with solutions having exactly one tip.

4.2.9 ezgraphGL.c

The changes compared to EZ-SPIRAL are mostly to the dialogue, which now recognizes several extra keys, and associated new functions.

4.2.10 ezgraphGL.h

There are no significant changes here compared to EZ-SPIRAL. This file contains a variety of macros used in the drawing routines of the spiral wave.

References

[1] D. Barkley. `http://www.maths.warwick.ac.uk/~barkley`.

[2] D. Barkley. A model for fast computer simulation of waves in excitable media. *Physica D*, 49:61–70, 1991.

[3] V. N. Biktashev, A. V. Holden, and E. V. Nikolaev. Spiral wave meander and symmetry of the plane. *International Journal of Bifurcation and Chaos*, 6(12A):2433–2440, 1996.

[4] A. J. Foulkes. *Drift and Meander of Spiral Waves*. PhD thesis, University of Liverpool, UK, 2009. arXiv:0912.4247, `http://www.csc.liv.ac.uk/~afoulkes/thesis.pdf`.