

Local Perturbation Analysis in MatCont

User's Guide & Tutorials

William R Holmes, May Anne Mata, Leah Edelstein-Keshet

October 13, 2014

Abstract

This document contains information about the **local perturbation analysis (LPA)**, also known as local perturbation analysis, and its application using MatCont, a graphical MATLAB package to study dynamical systems numerically. The goal of this paper is to give insight to the reader about the implementation of LPA in MatCont. We tackle two mathematical models namely wave-pinning and actin waves, and these will be used to illustrate the application of LPA using MatCont. Moreover, some technical issues are also presented here to show the advantages and disadvantages of these tools.

Contents

1	Introduction	3
2	Getting Started with MatCont	4
2.1	What is MatCont?	4
2.2	Installation and Launching MatCont GUI	4
2.3	Transcritical Bifurcation Example	6
2.3.1	Specifying the ODEs	6
2.3.2	Plotting details	7
2.3.3	Time Simulations	8
2.3.4	Equilibrium branch continuation	11
3	More Examples	17
3.1	Wave-pinning (WP) Model	17
3.1.1	Model description and LPA formulation	17
3.1.2	MatCont Steps	19
3.2	Actin Waves / (A,I,F) Model	24
3.2.1	LPA representation and model details	24
3.2.2	Implementation in MatCont	25
3.2.3	Two-parameter bifurcation diagram	25
4	Some Technical Difficulties	28
5	Plotting Enhancement	28

1 Introduction

Model analysis often requires tools to generate relevant sets of information that are necessary to understand a specific phenomenon. In biological systems, we often encounter reaction-diffusion equations. These can be studied using linear stability analysis and other dynamical system methods.

As with all PDE's, bifurcation analysis of reaction-diffusion systems is challenging. In this paper, we highlight a nonlinear stability analysis known as local perturbation analysis (LPA) that allows us to approximate the behaviour of a local pulse applied to the homogeneous steady-state. The model of the dynamics of local pulse is represented by a set of ODE's that are used for bifurcation analysis. A description of LPA will be presented in the next section.

This manual is designed for users who have background in bifurcation analysis but have not been introduced to LPA or MatCont. Though having MATLAB experience is an advantage, it is not required here. For researchers who are aspiring to study mathematical models in cellular systems, this document provides a good introduction of a cell polarization model and an intracellular actin waves model. These minimal models are used here for illustration purposes only. For in-depth discussion of these models, see [5, 6, 3, 2, 4].

2 Getting Started with MatCont

2.1 What is MatCont?

MatCont is a MATLAB-based software package developed under the supervision of W. Govaerts and Yu. A. Kuznetsov [1] for interactive numerical study of dynamical systems. This package has a number of capabilities and for the sake of this manual, we will only mention a few. (1) MatCont accesses MATLAB's ODE integrators and helps us integrate systems of ODEs without having to perform the actual MATLAB function calls. (2) It allows us to compute equilibrium solutions or fixed points to the system of ODEs and continue those equilibria with respect to a parameter of interest. And (3) it is able to detect bifurcations Hopf bifurcations, branch points, saddle-node bifurcations (or fold bifurcations), etc., which can be easily retrieved via graphical user interface (GUI) or command line application of the package. Since this bifurcation toolbox is MATLAB-based, it is easy to use, i.e. short learning curve, by those who have familiarity with MATLAB. Moreover, it is also platform independent in the sense that if you have learned how to use it on one platform, you can readily transport files and experience to other platform. The toolbox can be used with MATLAB on Windows, Unix, Linux, and MAC OSX. One must be warned however that running MatCont on MATLAB in OSX machine has limitations related to use of the C compiler. This along with a patch is mentioned on the notes that W.R. Holmes posted on this website: <http://www.math.ubc.ca/~7Ekeshet/MCB2012/wrholmes/MCB2012.html>.

2.2 Installation and Launching MatCont GUI

Download the package from <http://sourceforge.net/projects/matcont/> (google 'matcont') or go to the website <http://www.matcont.ugent.be/matcont.html>, which also contains installation details. Once the 'matcont4p2' (or 'matcont2.4' in the latter website) zipfile is downloaded, unzip the file and extract it in the directory of your choice. Start MATLAB and change your current working directory to the newly unzipped 'matcont' folder. Make sure that the m-file *matcont.m* is present in your 'matcont' folder otherwise the MatCont GUI will not be opened. To launch the GUI, type *matcont* at the command line. If you receive complaints about a compiler, type *mex-setup* in the command line and choose *lcc compiler*, then start MatCont again by typing *matcont*. For a detailed information about installing MatCont, please see the documentation on the above websites.

A snapshot of the GUI that pop-up after typing in *matcont* is displayed below.

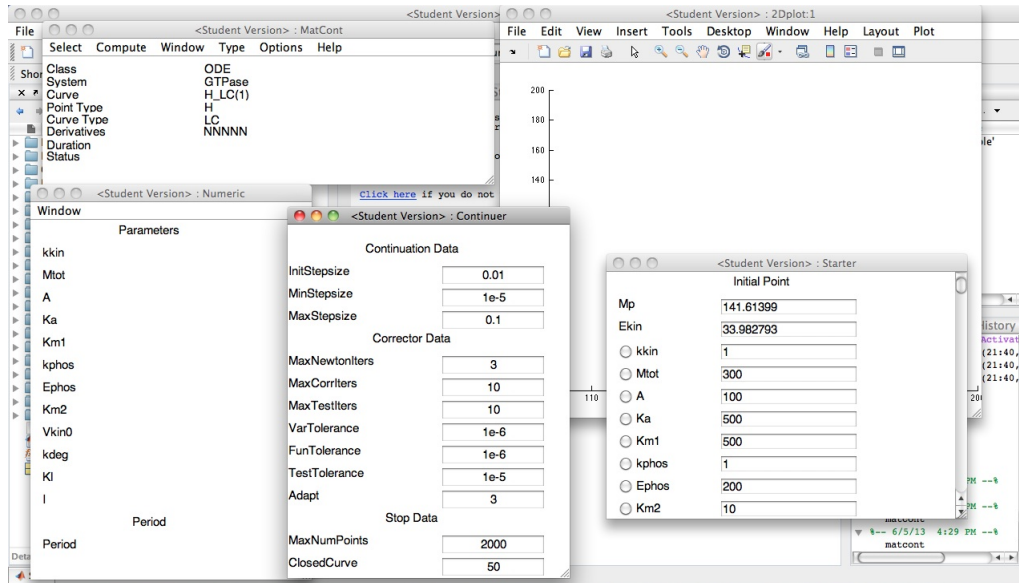


Fig. 3.2A: Graphical User Interface (GUI) of MatCont.

1. **MatCont** (upper left) – This is the main window where we can define the system, compute solutions, open plotting and numeric tools, set point and curve types, choose initial points, and see the status of our computation.
2. **Numeric** (lower left) – This window allows us to display current numerical values of the variables (also known as coordinates), parameters, eigenvalues, and other numerical data.
3. **Continuer** (middle) – This window enables us to adjust integration or continuation options that determine accuracy. It is noteworthy to mention that the most important parameter on this panel is 'MaxStepsize' which often needs to be changed to get a stable continuation.
4. **Starter** (lower right) – In this window, we can input initial values and choose the bifurcation parameter of interest.
5. **2Dplot** (upper right) – Depending on the aim of the study, this window can display a plot of the solution, phase portrait, and branch continuation curves. It also contains options that can change variables plotted and adjust plotting region.

In other versions of MatCont, windows 2-5 have to be opened manually from the options of window 1. The *continuer* and *starter* windows auto open when an operation is chosen while the *numeric* and *2Dplot* windows always require the user to open them when a new system is

initiated. For further instruction of opening the *2Dplot* window and plot setting descriptions, please refer to Section 2.3.2. Furthermore, the latter sections also describe how the *numeric* window be opened. For simplicity of this tutorial, we refer to these windows by their names in bold-italic face in either case.

2.3 Transcritical Bifurcation Example

As a simple example, let us consider the normal form of the transcritical bifurcation

$$\dot{x} = rx - x^2.$$

This model is commonly used in explaining fundamental concepts of bifurcation analysis for dynamical systems. It has a branch point at $r = 0$ that indicates emergence of two equilibrium curves. The two equilibria are $x = 0$ and $x = r$. When $r < 0$, the equilibrium branch $x = 0$, i.e. horizontal branch, is stable while $x = r$ is unstable. However, the two branches switch stability for $r > 0$. These features of transcritical bifurcation are verified as we plot it using MatCont.

2.3.1 Specifying the ODEs

We begin by specifying a new model via *MatCont*: **Select>New** as indicated in Fig. 3.3A.

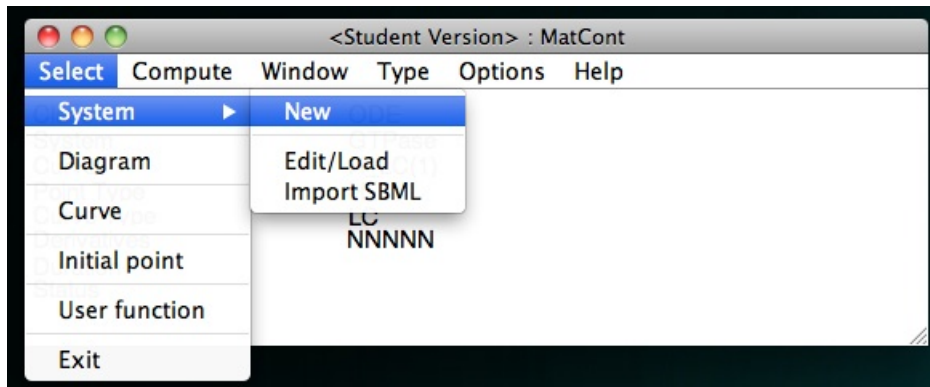


Fig. 3.3A: Choosing **Select>New** in *MatCont* menu.

A new window, named *System*, will immediately appear as displayed in Fig. 3.3B. This is where we input the ODE model. We have five text fields to be filled in:

1. Name – This is where we put an arbitrary name of the system or model e.g. ‘transcritical’.
2. Coordinates – The state variable/s of the system, e.g. ‘x’.
3. Parameter – This is where we write the parameter/s of the model, e.g. ‘r’.

4. Time – Naturally, we write ‘t’ here unless the system has a different independent variable.
5. Bigger box – This is where we input the ODE’s. We write $x'(t)$ as x' here and usual symbolic representation of MATLAB operations also applies.

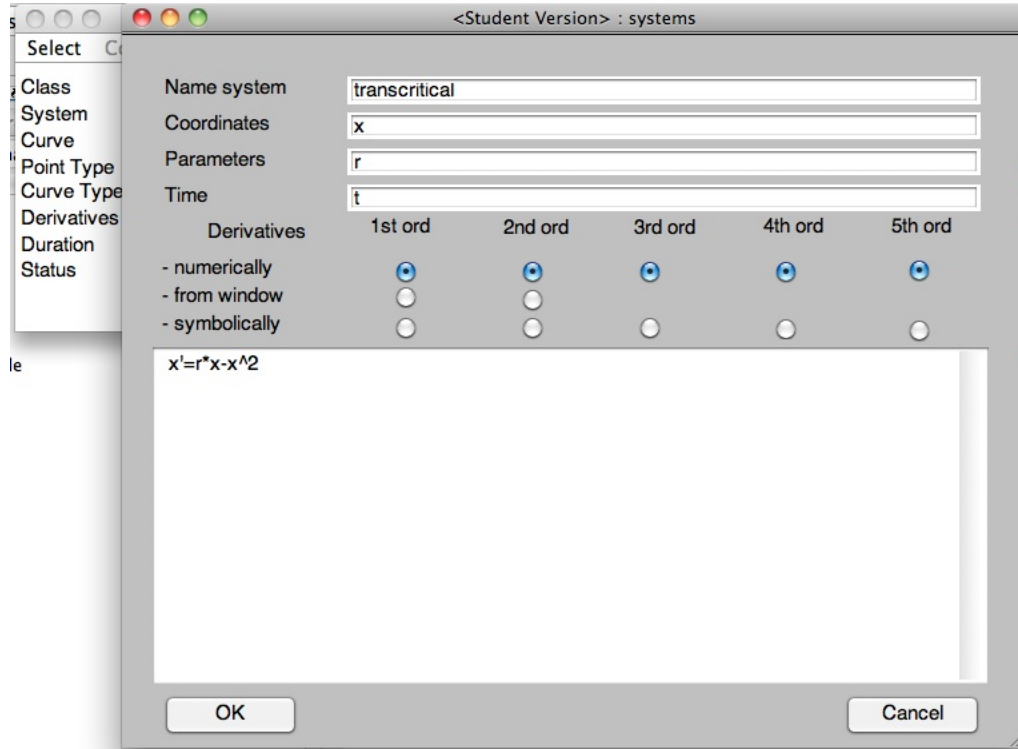


Fig. 3.3B: The *System* window.

Then, we click OK to validate that we are solving the system named ‘transcritical’. Once we click OK, *matcont* will display some information as shown below. Note that all other fields except for *Class*, *System*, and *Derivatives* are left blank. This means that we just started a fresh system and no computations have been done yet. Notice the ‘NNNNN’ displayed on *matcont* in 3.3C. This indicates that we chose the all the derivatives of the system to be computed numerically.

2.3.2 Plotting details

After specifying the model, we specify the plot setting. If you are interested in plotting the solution to the system, you can plot the profile by first launching the plot via *matcont*: **Window>Plot>2D-plot**. Then set the variables on the axes via *2Dplot*: **Layout>Variables on axes** (or **Attribute>Variables on axes**). You can also change the

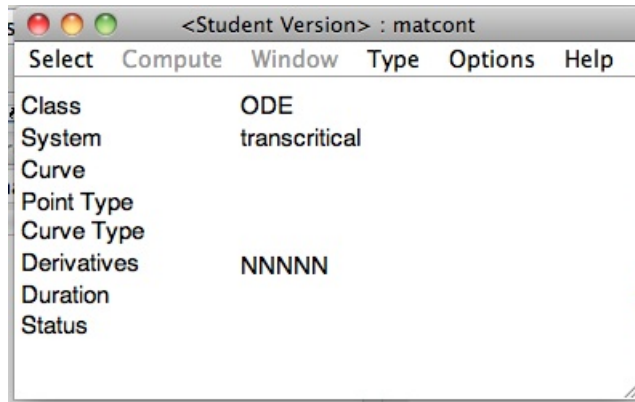


Fig. 3.3C: Appearance of *matcont* after we input the system.

range of each axis by going to **Layout>Plotting region**. Once the plot is set, you can start performing the necessary computations to solve the system.

2.3.3 Time Simulations

We can use MatCont to compute a solution to the system of ODEs. To initialize computation of an orbit (or solution) starting from initial values, we select **Type>Initial Point>Point** in *matcont* as in Fig. 3.3D.

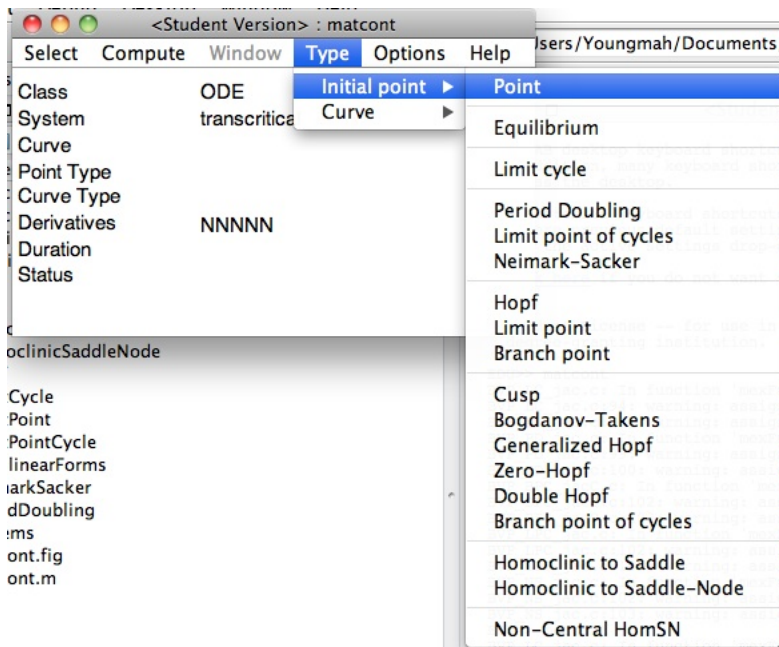


Fig. 3.3D: Specifying initial values with *matcont* menu.

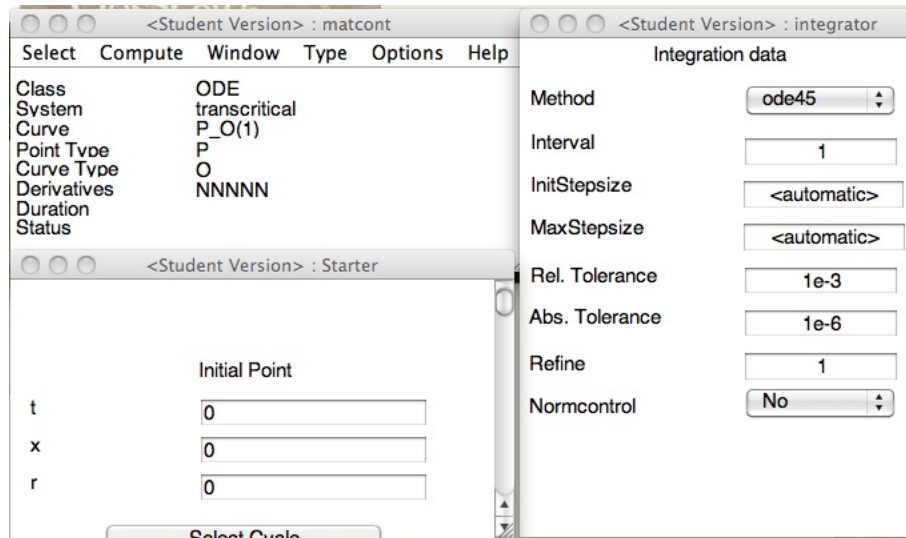


Fig. 3.3E: Altered setting in *matcont* with new windows named *Starter* (unadjusted) and *Integrator* after selecting **Type>Point>Initial Point**.

Two new windows will pop-up normally (See Fig. 3.3E): the *Starter* and *Integrator* windows. Notice as well that settings for the curve name and point/curve type were changed to ‘P_O’ in *matcont*. This indicates that we have set the initial point as a regular point (‘P’) and the curve as an orbit (‘O’). In *Starter*, we can input the initial values of the state variable/s (x) and parameter/s (r). A “Select Cycle” button is found in the same window. This is used for limit cycle continuation which is beyond the scope of this tutorial. Initially, let us choose $x = 1$ and $r = 0.5$ then set $t = 0$ (i.e. at time $t = 0$ the initial value 1 for the state variable and 0.5 for parameter).

On the other hand, *Integrator* shows us the integration data that we can vary. In particular, it allows us to change the type of ODE solver used to compute the solution of the system. You must refer to MATLAB’s help documentation to guide you on what solvers to use. Apparently, *ode45* is typically sufficient for non-stiff systems. The *Integrator* window also permits us to define the time interval (integration time) and other integration data that govern the details of the ODE solvers. Default parameters of this window usually work. Since we seek the steady state value of the state variable, we set the time interval (‘Interval’) to 10 from 1 in *Integrator* then we proceed to compute the solution forward in time. We do this via *matcont*: **Compute>Forward**. In case you want to display the plot of the solution, make sure to choose the appropriate axes (e.g. x for vertical axis, t for horizontal axis) and plotting region in *2dplot* before performing the computation. Otherwise, you will not be able to see the solution and may need to perform the computation again after setting a new plot. For the

chosen set of starting values, the solution is displayed on Fig. 3.3F .

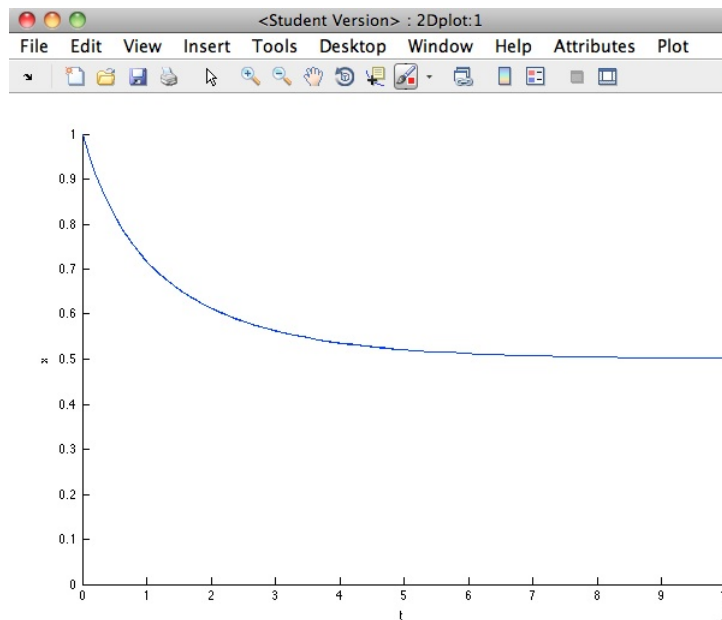


Fig. 3.3F: The solution to $\dot{x} = rx - x^2$ for $r = 0.5$ and $x(0) = 1$.
Vertical axis is x and the horizontal axis is t .

Notice that after forward computation, the settings in *matcont* were altered (See Fig. 3.3G). Under *Curve* name, we found 'P_O(1)' which indicates that we just had forward computation from the initial point. The point type 'P' means that the last point is a regular point while the curve type 'O' means that the collection of points forms an orbit or solution to the system. The word 'ready' means that MatCont has finished performing the computation and is ready for the next step.

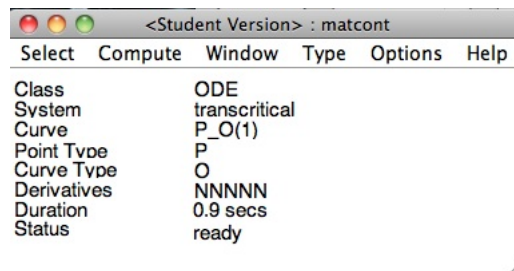


Fig. 3.3G: Status of *matcont* window after performing forward computation from an initial point.

2.3.4 Equilibrium branch continuation

Now we are ready to compute bifurcations of the system. Since we have found a solution to the system and saw it approach a steady-state value, we begin the equilibrium continuation via *matcont*: **Select>Initial Point**. This is done so we can choose the last point of the computed solution. A new window named *Select point* will appear showing the curve name ‘P_O(1)’ along with the first and last points of the orbit labeled as ‘00’ and ‘99’, respectively. We select the last point by hitting ‘2) 99: This is the last point of the orbit’ and pushing the “Select” button as indicated in Fig. 3.3H. Then we tell MatCont that this is an equilibrium point by selecting *matcont*: **Type>Initial Point >Equilibrium**.

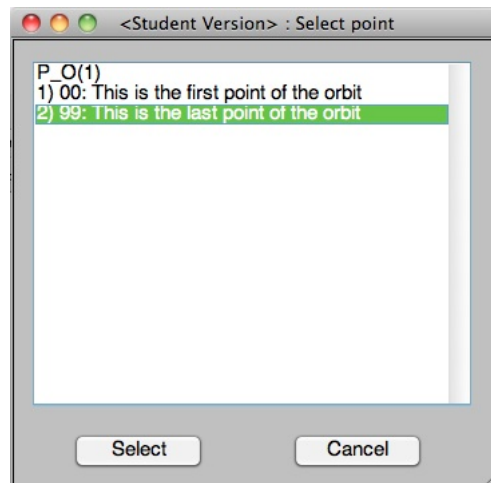


Fig. 3.3H: The Select Point window.

This leads to a new set of windows as shown in Fig. 3.3I. The displayed *Curve* name has changed to ‘EP_EP(1)’ with ‘EP’ (Equilibrium Point) as *Point* and *Curve types*. This means that the point chosen is assumed to be the equilibrium point and the curve we opt to draw is an equilibrium branch.

Before we compute the bifurcation points, we again set up the window for plotting. If *2Dplot* is initially launched, we set up the plotting region via *2dplot*: **Attributes>Variable on axes**. This brings up a new window that assigns variables for the plot as in Fig. 3.3J.

In this problem, we are interested in looking at how the equilibrium of our state variable x varies with respect to the parameter r . We also adjust the plotting region by going to **Attributes>Plotting region** where we can input the range of values for both axes as displayed on Fig. 3.3K. Here we assign it the value of (-1,1) for both axes. The modified *2dplot* window has appearance of Fig. 3.3L.

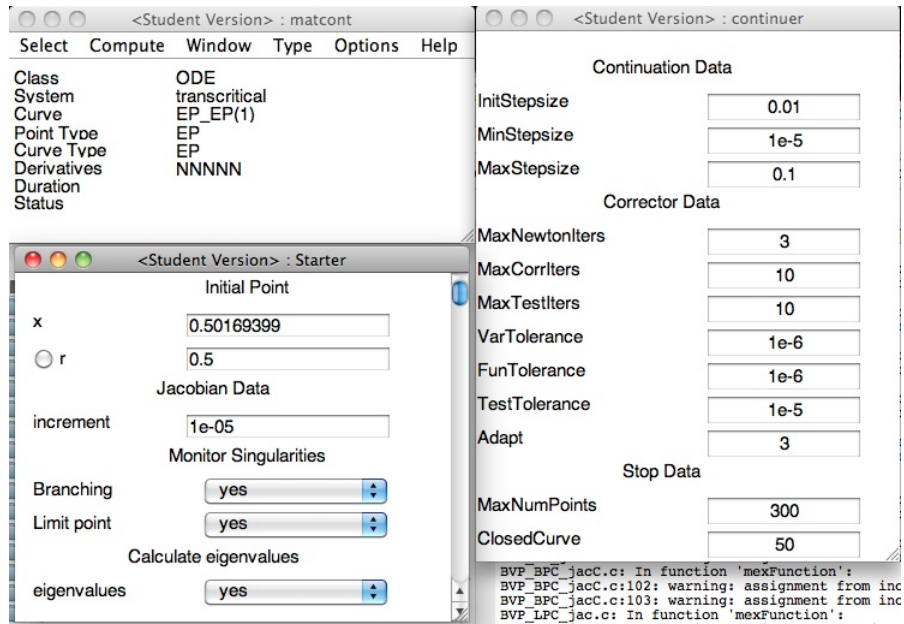


Fig. 3.3I: Settings of *matcont*, *starter*, and *continuer* windows as the equilibrium point is chosen.

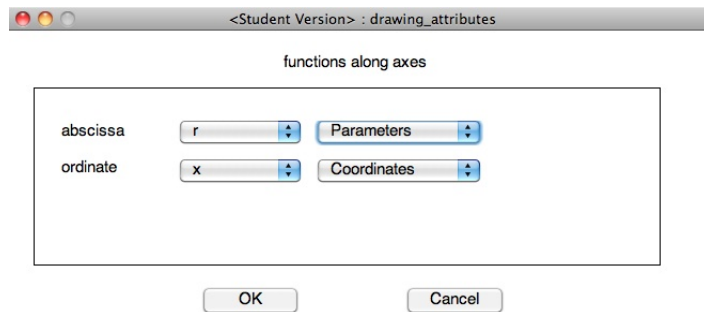


Fig. 3.3J: The *drawing attributes* window.

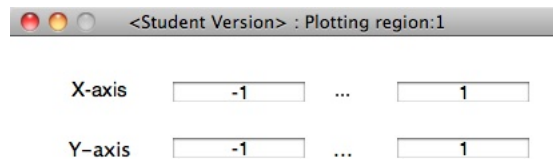


Fig. 3.3K: The *Plotting region* window.

On *Starter*, we tick the (radio)button beside *r* to indicate that it is our bifurcation parameter. Otherwise MatCont will prompt you to select a free parameter as you start your computation. The *numeric* window displays useful information related to the continuation.

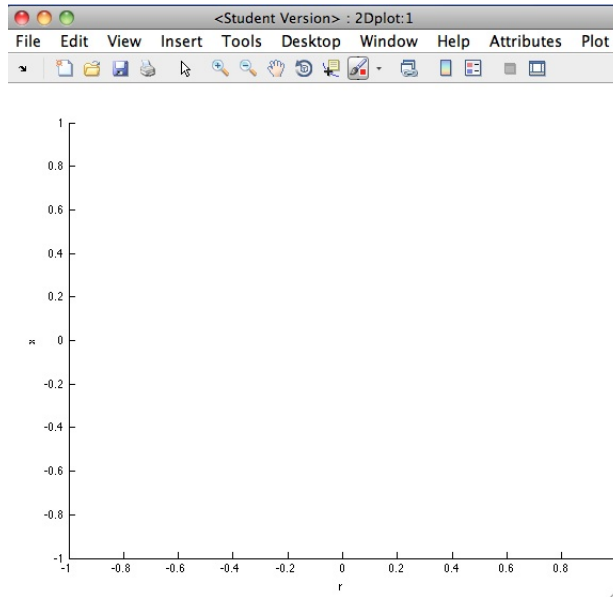


Fig. 3.3L: The adjusted *2dplot*. Vertical axis is the steady-state value of x and horizontal axis is r .

This can be shown by selecting *matcont*: **Window>Numeric** and then clicking on *numeric*: **Window>Layout**. A layout window will appear showing a list of data that can be retrieved. Here we choose ‘eigenvalues’. When this is selected, it is shown as upper case. Then, click OK. Maximizing the numeric window helps see the ‘Eigenvalues’ feature that keeps track of the real and imaginary part of the eigenvalues associated with the Jacobian of the system. These eigenvalues determine the stability of the equilibrium branch. These steps are portrayed in Fig. 3.3M.

Continue the equilibrium via *matcont*: **Compute>Forward**. Observe that a tiny window will pop-up that helps to control the continuation and the status becomes ‘computing’. We can let it compute forward until it stops or we can opt to stop it by pushing the ‘Stop’ button in the tiny window displayed in Fig. 3.3N. The resulting plot is shown in the same figure. Since we want to see how the branch behaves for a chosen range of r , we can also do a backward computation (**Compute>Backward**) for equilibrium branch continuation. Fig. 3.3O shows the results of this continuation.

Notice that we have a Branch Point labelled as ‘BP’ found on equilibrium curve. This indicates that another equilibrium branch emanates from this point. To draw the other branch (or perform an equilibrium branch continuation from a branch point), we do the following:

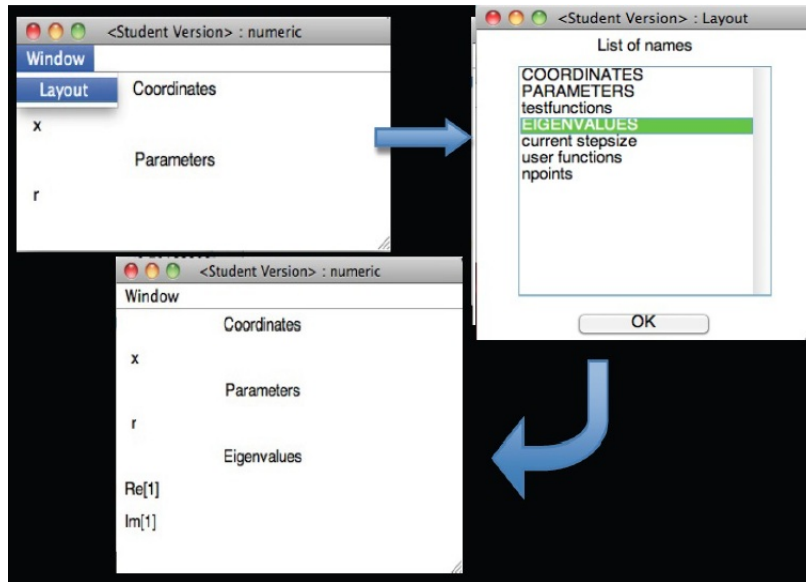


Fig. 3.3M: Transitions in *numeric* window as we choose to display ‘Eigenvalues’ associated with the system.

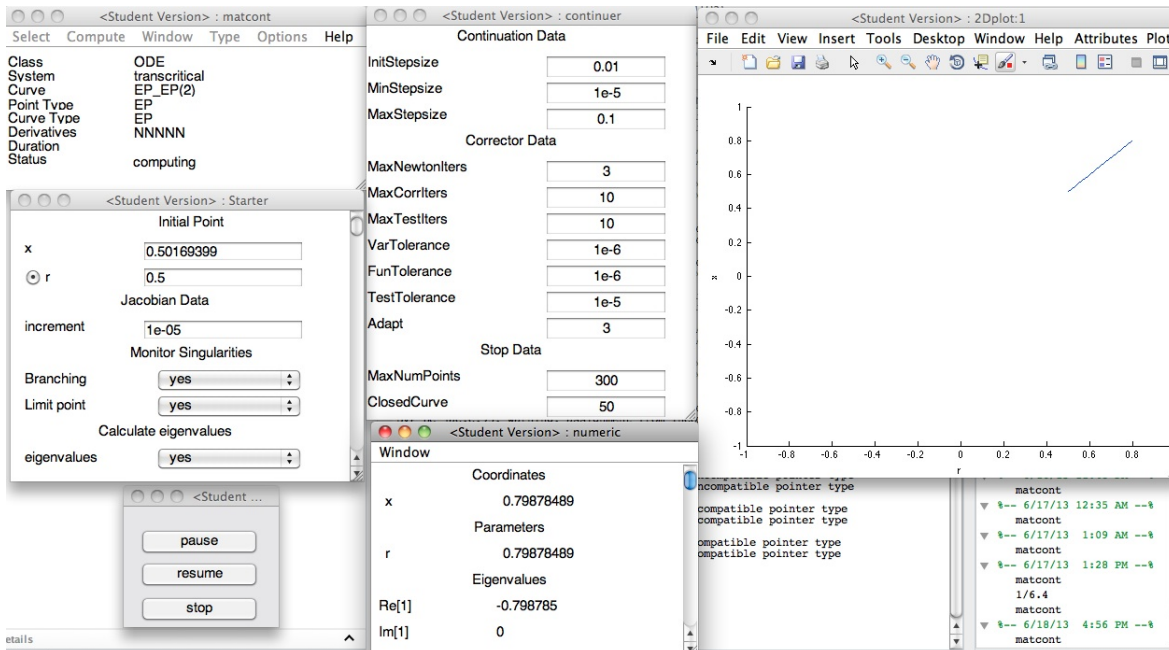


Fig. 3.3N: Forward computation for equilibrium branch continuation.

1. *matcont*: Select>Initial Point
2. Choose ‘BP’ under EP_EP(1) or EP_EP(2).
3. Then click the “Select” button

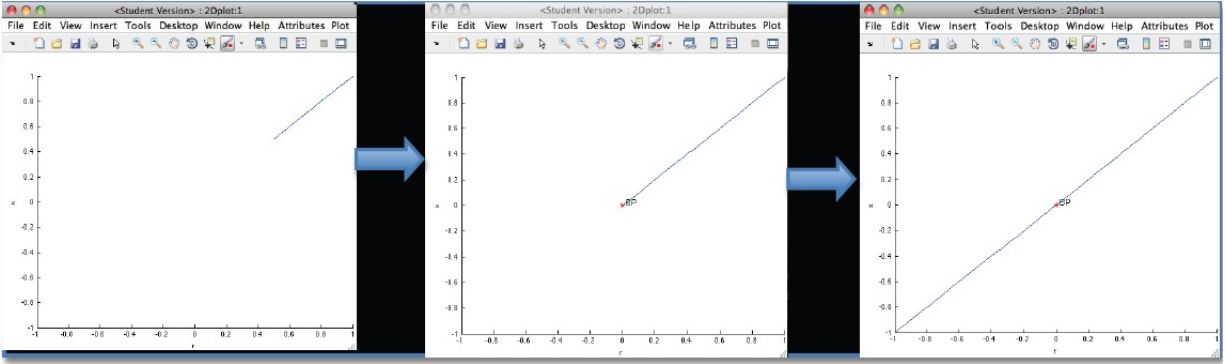


Fig. 3.3O: Traces of equilibrium branch as backward computation is performed.

Then, *matcont* will display a different setting as shown in Fig. 3.3P. Notice the *Curve* name ‘BP_BP(1)’ which means that the point and curve types are currently assigned as branch points.

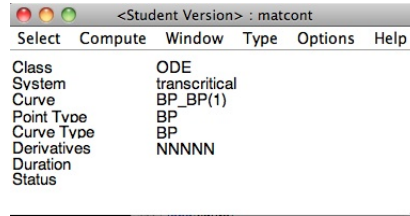


Fig. 3.3P: The setting of *matcont* after we set the branch point as initial point.

However, we wish to draw an equilibrium curve from the chosen branch point. Thus, we need to make sure that the curve type must be assigned as ‘EP’. We can do this via *matcont*: **Type>Curve>Equilibrium** which yields us Fig. 3.3Q.

Finally, we compute forward (backward) by *matcont*: **Compute>Forward (Backward)**. As seen in Fig. 3.3R, the one-parameter bifurcation plot of $\dot{x} = rx - x^2$ shows fixed points that exist for all values of r . Moreover, the stability of such fixed points interchanges as the parameter is varied. You can check from the eigenvalues that the equilibrium branches have different stability to the left and to the right of the bifurcation point. In particular, if $r < 0$, the horizontal branch is stable while the other one is unstable; the stability for each branch switches when $r > 0$. This bifurcation structure depicts the canonical transcritical bifurcation.

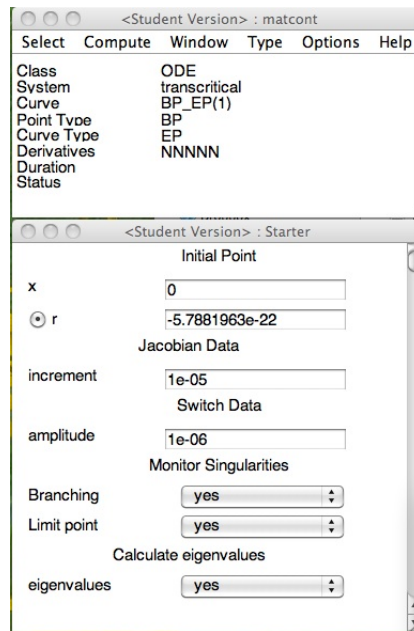


Fig. 3.3Q: Display of *matcont* and *Starter* after we set the branch point as initial point and curve type to equilibrium.

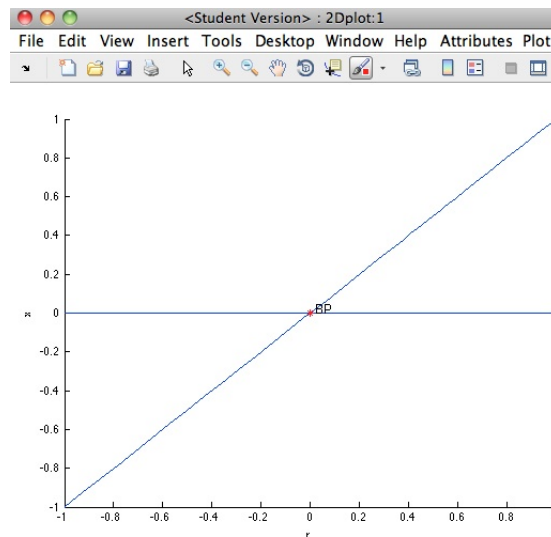


Fig. 3.3R: MatCont-generated transcritical bifurcation diagram.

3 More Examples

In this section, we consider examples of RD models of cellular system, their LPA formulation and investigation in MatCont. First, we consider a wave-pinning (WP) model that has been found to be a minimal model for cell polarization (asymmetrical distribution of the polarity components along the domain of the cell)[5]. As a second example, we consider a model of actin wave formation and propagation based on WP model with an additional component in the circuit, the amount of filamentous actin (F-actin). We consider a 1D cell spatial domain where x is position along a ‘cell diameter’. No-flux boundary conditions are assumed. The reader is referred to the works of [5] and [2] for a detailed model development. Schematic representations of the models are presented in Fig. 4.

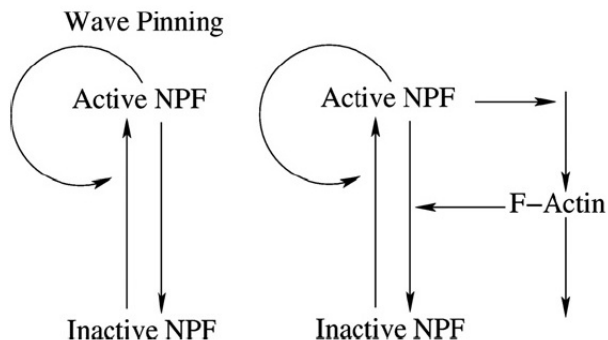


Fig. 4: Schematic diagram for (left) wave-pinning (WP) and (right) actin waves (A,I,F) model [2].

3.1 Wave-pinning (WP) Model

3.1.1 Model description and LPA formulation

The minimal wave-pinning model for cell polarization involves a regulatory protein, belonging to a family of small GTPases. These proteins have two states, active and inactive with autocatalytic active component. See Fig. 4. Let u and v be the amount of membrane-bound (active form) and cytosolic (inactive form) small GTPase, respectively. Cycling of GTPase between active and inactive state takes place while the total amount is conserved. A Hill function is used to represent autocatalytic positive feedback on u . We also assume here that the active component is slow-diffusing and the inactive component is fast-diffusing, i.e. $D_u \ll D_v$. Using reaction-

diffusion equations, the WP model has the form:

$$\frac{\partial u}{\partial t} = f(u, v) + D_u \frac{\partial^2 u}{\partial x^2}, \quad (1a)$$

$$\frac{\partial v}{\partial t} = -f(u, v) + D_v \frac{\partial^2 v}{\partial x^2}, \quad (1b)$$

with a simplified set of kinetics is given by: $f(u, v) = \left(k_0 + \frac{\gamma u^n}{u_0^n + u^n} \right) v - \delta u$ as explained in [5].

From the kinetics above, k_0 represents the basal activation rate, which is chosen as the bifurcation parameter. The parameter γ controls the strength of the positive feedback by the active component. The sharpness of the response is determined by the Hill coefficient n while u_0 is a reference value of the active component at which autocatalysis is ‘turned on’. The model has a travelling wave solution in response to a heterogeneous stimulus a wave that slows as it traverses the domain until it stalls (velocity 0) in the interior, leaving a polarized state. The large difference in diffusion motivates us to use LPA to explore the behaviour of the homogeneous steady state (HSS) over a range of a parameter of interest. Note that Eqs. (1) satisfy the assumption for LPA. We can then readily formulate the corresponding LPA-ODEs as follows:

$$\frac{du_l}{dt} = f(u_l, v_g), \quad (2a)$$

$$\frac{du_g}{dt} = f(u_g, v_g), \quad (2b)$$

$$\frac{dv_g}{dt} = -f(u_g, v_g). \quad (2c)$$

It is clear that u is considered to be our slow variable and so in LPA, it has equations describing both in ‘local’ and ‘global’ forms (represented by subscript l and g , respectively). On the other hand, v is simply regarded as a ‘global’ form since it is fast diffusing. Note that $u_g + v_g = T$, the total average concentration, is constant and so we can use this fact to eliminate one variable. The reduced system of LPA-ODEs is:

$$\frac{du_l}{dt} = \left(k_0 + \frac{\gamma u_l^n}{u_0^n + u_l^n} \right) (T - u_g) - \delta u_l, \quad (3a)$$

$$\frac{du_g}{dt} = \left(k_0 + \frac{\gamma u_g^n}{u_0^n + u_g^n} \right) (T - u_g) - \delta u_g. \quad (3b)$$

The LPA-ODEs (2) have a zero eigenvalue that comes directly from conservation property. The substitution done earlier is necessary to remove a degenerate eigenvalue of the Jacobian of

the LPA system, which will cause fatal errors in any continuation software. Hence, we use Eqs. (3) instead of (2) in order to avoid errors due to a zero eigenvalue, e.g. ‘No convergence at x0’, related to eigenvalue computation in a singular Jacobian matrix. Furthermore, the parameter values $\gamma = u_0 = \delta = 1$, $T = 2.27$, and $n = 3$ are used to compute the solution and bifurcation points of Eqs. (3) in MatCont.

3.1.2 MatCont Steps

Here we will briefly discuss the steps to perform the analysis of Eqs. (3) using MatCont. Each step has embedded figure that pictorially describes operations or actions. You can refer to the figures as quick guide to the step.

Step 1. Define the system and compute the solution. (Fig. 4.1A). This step specifies a new system called ‘wavepinning’ and runs a time simulation of Eqs. (3) to find an orbit starting from an initial point. For clarity, the input information in *systems* window are provided below:

1. Name system: *wavepinning*
2. Coordinates: *uloc,uglo*
3. Parameters: *k0,u0,gam,T,del,n*
4. Time: *t*
5. $uloc' = (k0 + gam * uloc^n / (u0^n + uloc^n)) * (T - uglo) - del * uloc$
 $uglo' = (k0 + gam * uglo^n / (u0^n + uglo^n)) * (T - uglo) - del * uglo$

Step 2. Global/well-mixed equilibrium branch continuation. (Fig. 4.1B). In [2], it was shown that the LPA bifurcation for WP model has an interesting feature. It possesses two equilibrium branches that are identified as global and local branches. The global/well-mixed equilibrium branch refers to the equilibrium solution where the steady-state values of u_l is equal to u_g . This is also the homogeneous steady state associated with a well-mixed version of Eqs. (1) (no diffusion term). In this step, you will learn how this global branch can be generated in MatCont from the highlighted sequence of moves. Notice that the branch has two branch points (BP) and two neutral-saddles (H). In the study of LPA for WP model, only the BPs are taken into consideration as they mark changes in the stability of the equilibrium branch.

Step 3. Local equilibrium branch continuation. (Fig. 4.1C). Next, we complete the LPA bifurcation plot by doing a local equilibrium branch continuation. The sequence of

actions is described in Fig. 4.1C. As shown here, another special point appeared labelled ‘LP’ (for limit point or fold-point bifurcation). Here we expect a change in the stability of the local branch as confirmed by signs of the real parts of all eigenvalues.

The resulting LPA bifurcation plot gives us insight about how the local pulse behave as we vary the parameter k_0 . For instance, consider the regime from $k_0 = 0$ to the first *BP*. The well-mixed branch is stable while there is an unstable local branch (middle) that acts as threshold: a pulse above this threshold will grow and be attracted to a higher-steady state (stable local branch). We refer to this as the wave-pinning (WP) regime. See [2] for thorough interpretation of the plot.

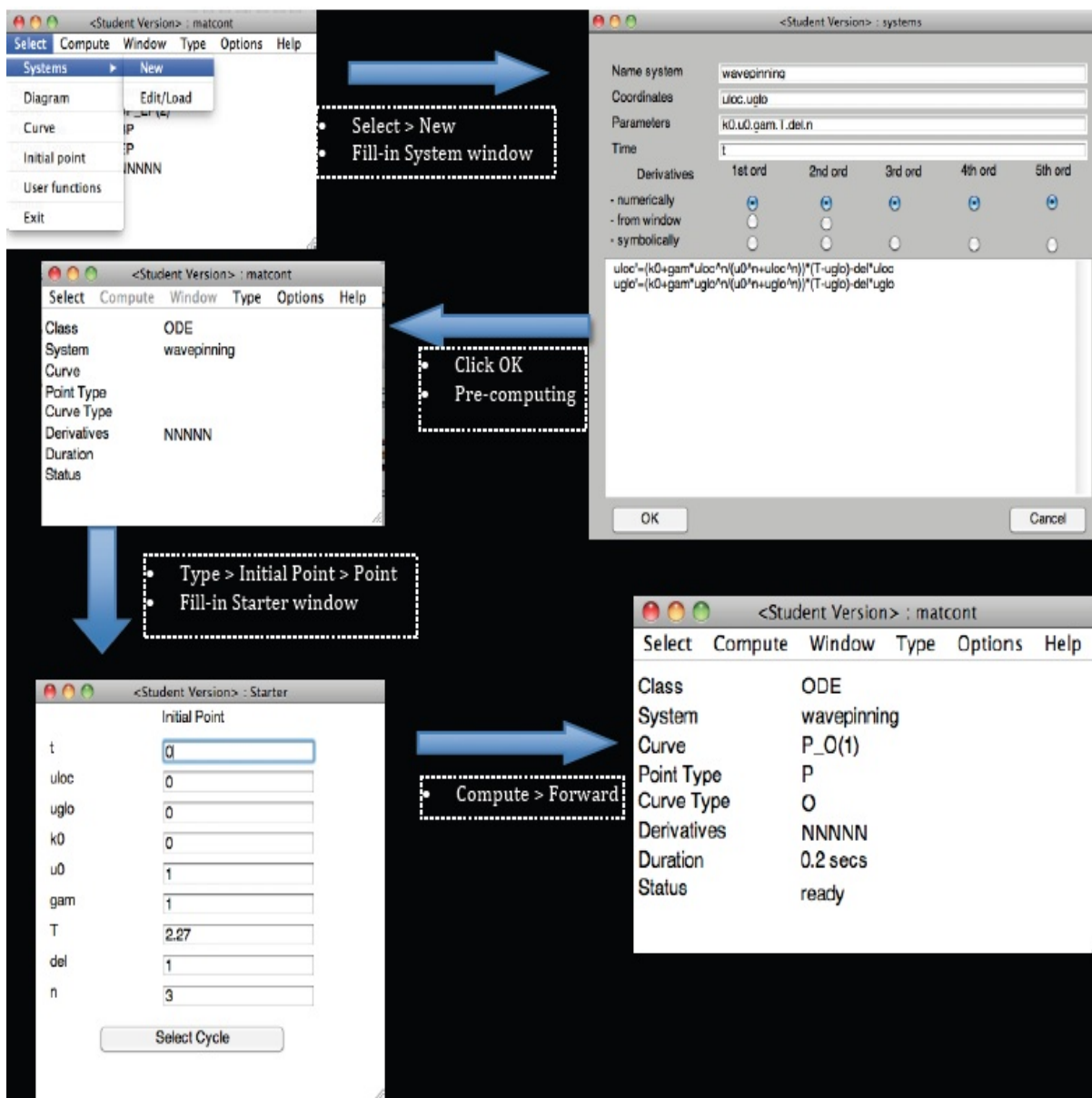


Fig. 4.1A: Sequence of operations executed and outputs made as we define the system and compute its solution.

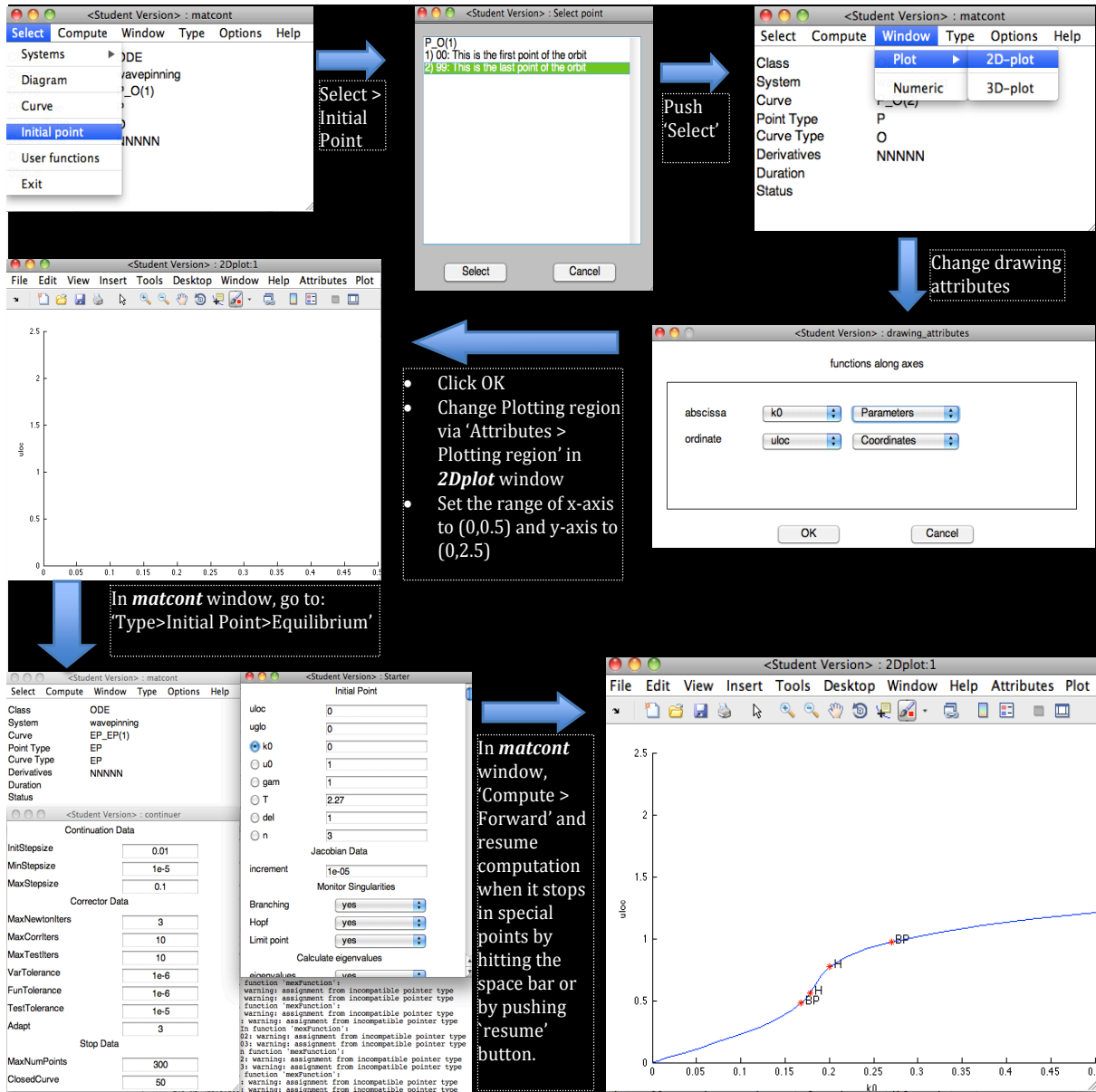


Fig. 4.1B: Sequence of actions to perform a global/well-mixed equilibrium branch continuation.

3.2 Actin Waves / (A,I,F) Model

3.2.1 LPA representation and model details

We consider another model for cellular systems describes the formation and propagation actin waves. This model was developed to answer questions about a phenomenon found in a eukaryotic cell where waves of actin, a cytoskeletal component, propagate along a substrate attached membrane [8, 7]. Holmes et al. [2, 4] modeled this by extending the WP module. The model is composed of three components: active and inactive form of regulatory protein known as nucleation promoting factors (NPF), and filamentous-actin (F-actin). The state variables of the model are the amounts of active NPF (A), inactive NPF (I), and F-actin (F) whose interactions are described in Fig. 4. The active NPF and F-actin are considered to be slow-diffusing because they are membrane associated. On the other hand, inactive NPF is considered fast-diffusing. From [2], the actin waves (A,I,F) model is given by:

$$\frac{\partial A}{\partial t} = f(A, I, F) + D_A \frac{\partial^2 A}{\partial x^2}, \quad (4a)$$

$$\frac{\partial I}{\partial t} = -f(A, I, F) + D_I \frac{\partial^2 I}{\partial x^2}, \quad (4b)$$

$$\frac{\partial F}{\partial t} = \varepsilon h(A, F). \quad (4c)$$

with kinetics terms:

$$f(A, I, F) = \left(k_0 + \gamma \frac{A^3}{A_0^3 + A^3} \right) I - \left(s_1 + s_2 \frac{F}{1 + F} \right) A, \quad h(A, F) = k_n A - k_s F.$$

The LPA representation of the (A,I,F) model then has the following form:

$$\frac{dA_l}{dt} = f(A_l, I_g, F_l) \quad (5a)$$

$$\frac{dA_g}{dt} = f(A_g, I_g, F_g) \quad (5b)$$

$$\frac{dI_g}{dt} = -f(A_g, I_g, F_g) \quad (5c)$$

$$\frac{dF_l}{dt} = \varepsilon h(A_l, F_l) \quad (5d)$$

$$\frac{dF_g}{dt} = \varepsilon h(A_g, F_g) \quad (5e)$$

Again, we utilize the conservation property $A_g + I_g = T_{NPF}$ to eliminate Eq. (5)c as we implement the set of LPA-ODEs in MatCont.

3.2.2 Implementation in MatCont

Following the methods introduced in the previous section, making a one-parameter bifurcation diagram of A_l against k_0 is straightforward. At this point, we assume that you are now familiar with how MatCont works as a bifurcation tool and so we will only enumerate here the general steps to compute bifurcation points for the (A,I,F) model. Parameter values to use are: $\gamma = 1$, $A_0 = 0.4$, $s_1 = 0.8$, $s_2 = 0.7$, $T_{NPF} = 1$, $\varepsilon = 0.1$, $k_n = 2$, $k_s = 0.25$. Initial values of $A_{l,g}$, $F_{l,g}$, and k_0 are all set to zero. The steps to generate the bifurcation plot, as displayed in Fig. 4.2A, in MatCont are as follows:

1. Specify the new system ‘actinwave’ via **matcont: Select>New**.
2. Perform a time simulation of the system to obtain its solution from initial values of variables and parameters.
3. Take the last point of the solution as the equilibrium.
4. Prepare the bifurcation plot by launching the **2Dplot** window.
5. Change the ‘MaxStepSize’ in **Contiuer** to 0.01. This allows us to capture the Hopf bifurcation points (also labelled as ‘H’). You can distinguish a Hopf point from a Neutral-Saddle point by means of the ‘status’ displayed on **matcont** as we perform equilibrium continuation. Alternatively, one can also check MATLAB’s command window to see the coordinates of the special point.
6. Do equilibrium branch continuation. *Warning: Computations may stop without completing the branch. If this happens, you can extend the computation via **matcont: Comput>Extend**.*

3.2.3 Two-parameter bifurcation diagram

Sometimes we want to do a two-parameter bifurcation analysis of the system to see its qualitative behavior as we vary two parameters. This can also be done in MatCont. We begin by preparing the plot. In [2], the actin wave model is studied further by considering the effect of k_0 and s_2 on the solution of the system. Thus, we first set the variables on the axes of our plot as k_0 (abscissa) and s_2 (ordinate) via **2dplot: Layout>Variables on axes**. If we want to keep track of the Hopf points for different value of k_0 and s_2 , we do the following:

1. Choose the first Hopf point in the global branch via **matcont: Select>Initial Point** under either ‘EP_EP(1)’ or ‘EP_EP(2)’ curve name.

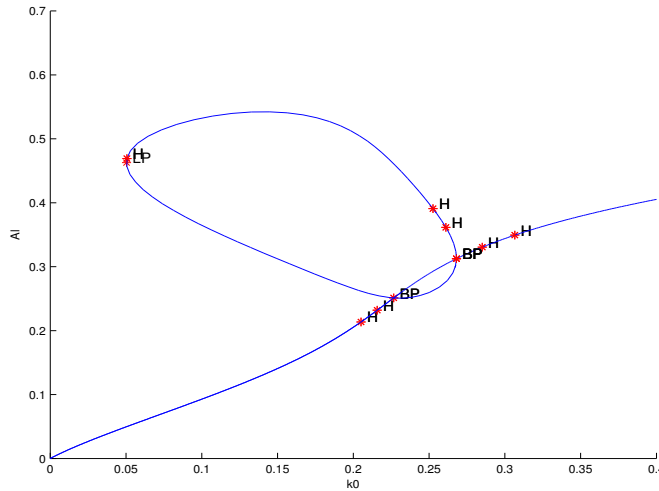


Fig. 4.2A: LPA bifurcation diagram for the actin waves model given by Eqs. (5a,b,d,e).

2. Notice that the curve name in *matcont* changes to ‘H_LC(1)’, which suggest that the point type is a Hopf point and the curve type is a limit cycle.
3. Since we want to trace a Hopf curve, the curve type must be set to ‘Hopf’. We do this by selecting **Type>Curve>Hopf** in *matcont*.
4. Do *matcont*: **Compute>Forward** to draw the Hopf curve in the two-parameter bifurcation plot. For the full plot, also do *matcont*: **Compute>Backward**.

The Hopf curve looks like this:

We can also keep track of the branch points, as displayed in Fig. 4.2C by doing the following:

1. Choose the first branch point in the global branch via *matcont*: **Select>Initial Point** under either ‘EP_EP(1)’ or ‘EP_EP(2)’ curve name
2. Notice that the curve name in *matcont* window becomes ‘BP_BP(1)’, which suggest that the point and curve types are both branch point.
3. To trace a branch point curve (BP curve), the curve type must be set to ‘Limit Point’. We do this via *matcont*: **Type>Curve>Limit Point**.
4. Finally, we do *matcont*: **Compute>Forward** to draw the BP curve in our two-parameter bifurcation plot. Similarly, we perform *matcont*: **Compute>Backward** for a full plot.

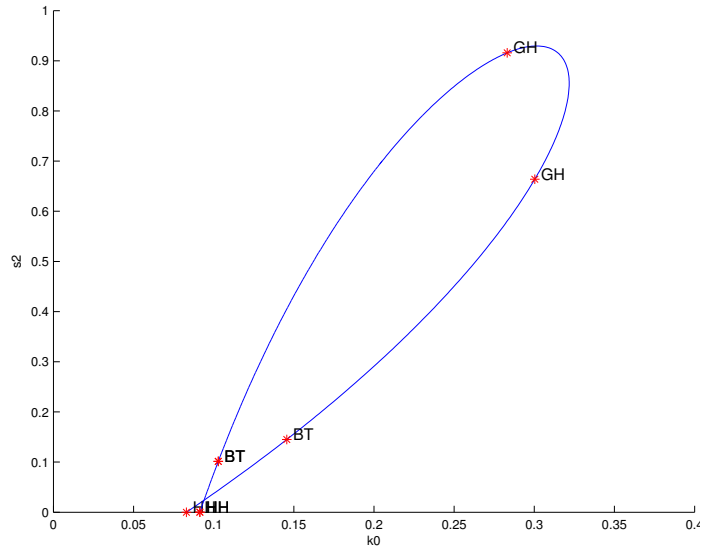


Fig. 4.2B: The Hopf curve in the k_0 - s_2 bifurcation plot for (A,I,F) model.

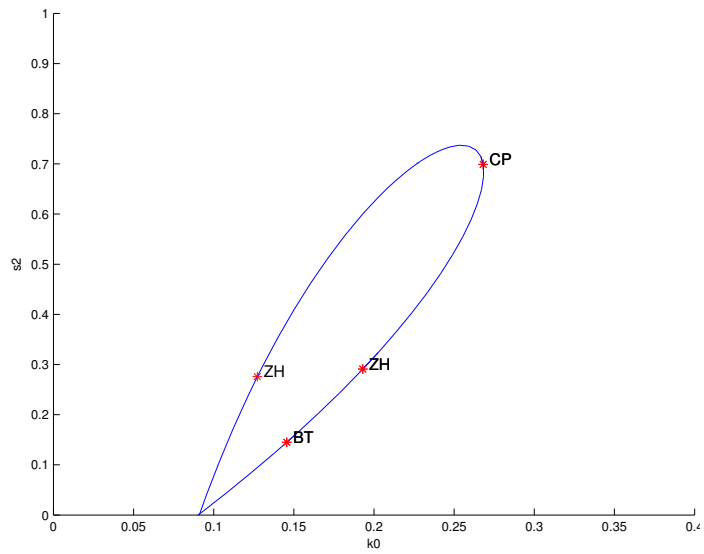


Fig. 4.2C: The branch point curve in two-parameter bifurcation plot for (A,I,F) model.

4 Some Technical Difficulties

A few technical issues are often encountered in MatCont. We mentioned in Section 2 the solution to the error regarding missing compiler when starting MatCont. Another possible mistake is the use of the character ‘c’ (‘C’) as variable or parameter label when specifying the system. Using ‘c’ as label confuses MATLAB since it is used as cell array assignment and not as variable or parameter name. The remedy for this is easy: *avoid using ‘c’ as variable or parameter name.*

Sometimes when we perform an equilibrium continuation, an error “No convergence at x0” is obtained. This may mean that the system either blows up or is discontinuous. To fix this, try different initial conditions. It is important to initiate the continuation very close to an equilibrium point. Trying a different *ode* solver also works. Or else, increase the time interval for simulation, i.e. choose a larger number in the “Time Interval” field under *Integrator*, since it could be that the last point of the orbit is not close enough to the fixed point. Furthermore, you may also encounter an error saying “Current step size is small”. This can be fixed by varying “MinStepSize” and “MaxStepSize” in the *continuer* window.

So far these are the common errors using MatCont as a bifurcation tool. For other errors and technical problems, look for an answer online with google. We have expanding documentation on MatCont’s fixed bugs and errors as the number of MatCont users is increasing.

5 Plotting Enhancement

As you can see, the plots alone do not provide information about the stability of the equilibrium branches. To check stability, one must determine the eigenvalues by eye as the computation runs or look at the output file under *Systems* folder. Doing this is tedious and cumbersome.

Fortunately, we can perform a MatCont computation using command line driver that allows us to customize the plot. You can study how the plot is enhanced using the MATLAB codes found in <http://www.math.ubc.ca/~keshet/MCB2012/wrholmes/MCB2012.html>. The Matcont Function contains the function file associated with the input system and the Command Line Driver shows the main file. Take the “Wave Pinning Codes” for example. Download the associated MATLAB files and put them in a folder called *Cl.matcont4p2*. Before running the main file, you should first run *init.m* to access the functions stored in other directories of *Cl.matcont4p2*. Then, run the main file, *WavePinningGL_Cont_Driver1.m*, that simply generates a nicer bifurcation plot found in Fig. 6.

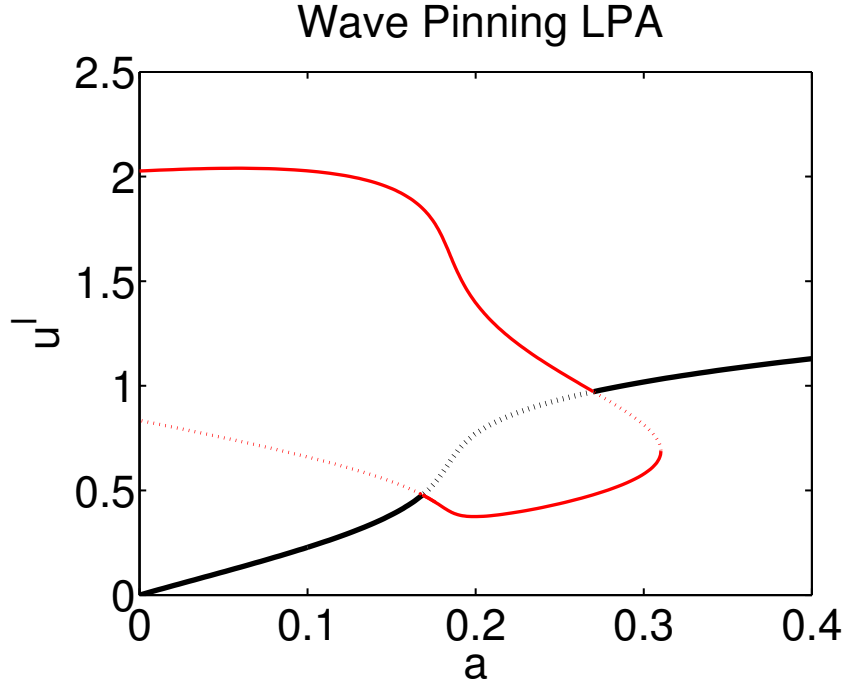


Fig. 6: The enhanced bifurcation plot for wave-pinning model. Black and red curves represent global and local solution branches respectively.

References

- [1] A. DHOOGHE, W. GOVAERTS, AND Y. KUZNETSOV, *Matcont: A Matlab package for numerical bifurcation analysis of ODEs*, ACM TOMS, 29 (2003), pp. 141–164.
- [2] W. HOLMES, A. CARLSSON, AND L. EDELSTEIN-KESHET, *Regimes of wave type patterning driven by refractory actin feedback: transition from static polarization to dynamic wave behaviour*, Physical Biology, 9 (2012), p. 046005.
- [3] A. JILKINE AND L. EDELSTEIN-KESHET, *A comparison of mathematical models for polarization of single eukaryotic cells in response to guided cues*, PLoS Comput Biol, 7 (2011), p. e1001121.
- [4] M. A. MATA, M. DUTOT, L. EDELSTEIN-KESHET, AND W. R. HOLMES, *A model for intracellular actin waves explored by nonlinear local perturbation analysis*, Journal of Theoretical Biology, 334 (2013), pp. 149–161.
- [5] Y. MORI, A. JILKINE, AND L. EDELSTEIN-KESHET, *Wave-pinning and cell polarity from a bistable reaction-diffusion system*, Biophysical Journal, 94 (2008), pp. 3684–3697.

- [6] ———, *Asymptotic and bifurcation analysis of wave-pinning in a reaction-diffusion model for cell polarization.*, SIAM J Applied Math, 71 (2011), pp. 1401–1427.
- [7] M. VICKER, *Eukaryotic cell locomotion depends on the propagation of self-organized reaction-diffusion waves and oscillations of actin filament assembly*, J. Expt. Cell Res., 275 (2002), pp. 54–66.
- [8] ———, *F-actin assembly in dictyostelium cell locomotion and shape oscillations propagates as a self-organized reaction-diffusion wave*, FEBS Lett., 510 (2002), pp. 5–9.