Stewart Heitmann
Michael Breakspear

Handbook for the Brain Dynamics Toolbox

Version 2018a

SAMPLE CHAPTER

Computational Neuroscience

QIMR Berghofer Medical Research Institute

Stewart Heitmann
Michael Breakspear

Handbook for the **Brain Dynamics Toolbox**

Version 2018a

Computational Neuroscience

QIMR Berghofer Medical Research Institute

Stewart Heitmann
QIMR Berghofer Medical Research Institute
300 Herston Road, Herston QLD 4006, Australia
Stewart.Heitmann@qimrberghofer.edu.au

Michael Breakspear
QIMR Berghofer Medical Research Institute
300 Herston Road, Herston QLD 4006, Australia
Michael.Breakspear@qimrberghofer.edu.au

http://www.bdtoolbox.org

Handbook for the Brain Dynamics Toolbox: Version 2018a. QIMR Berghofer Medical Research Institute. Paperback ISBN 9781980572503.

Cover artwork is a detail from *Fone Books* by Stewart Heitmann, 2001. Acrylic paint on board. Digitally enhanced colours.

Preface

Dynamical systems have been fundamental to theoretical neuroscience ever since Alan Hodgkin and Andrew Huxley first elucidated the dynamics of spiking neural membranes [21] in the 1950s. Computational neuroscience has progressed enormously since those early days but non-linear differential equations remain at its core. In practice, such equations can only be solved numerically in all but a few special cases. Numerical methods have thus become an integral part of computational neuroscience and software toolkits are the manifestation of those endeavours. Many toolkits have been developed over the years. Each one represents an attempt to balance the shifting tensions between mathematical flexibility and computational convenience. Early toolkits such as GENESIS [2], NEURON [6] and BRIAN [16] focussed on conductance-based models to simulate small networks of neurons with great biological detail. The more contemporary Virtual Brain [37] scales up that approach to the level of the whole brain. It combines the dynamics of large-scale neural populations with detailed biological networks (connectomes) to simulate realistic EEG, MEG and fMRI signals. It focuses on the role of neuronal connectivity in healthy and diseased brains. At the mathematical level, dynamical systems toolkits such as AUTO [11], XPPAUT [12], MATCONT [10], PyDSToolkit [7] and CoCo [9] provide advanced methods for analysing the steady-state behaviour of non-linear dynamical systems. These toolkits are useful for analysing the bifurcation structure of neural dynamics but often assume a substantial background in mathematical theory.

Despite this existing corpus of toolboxes, our experience of publishing and teaching computational neuroscience suggests there exists a gap that links theory to practise. In particular, students trained in the varied disciplines of neuroscience often struggle with translating their considerable interest into computational research. This is particularly true in the fledging fields of computational cognitive and neuroimaging science. The Brain

Dynamics Toolbox is designed to bridge this gap, allowing those with diverse backgrounds and an interest in neuronal dynamics to explore a variety of models through phase space analysis, time series exploration and other analytic methods. The design of the graphical interface fosters an educational experience, yet there is no barrier to scripting large-scale simulations and surveys. The toolbox thus fills the gap between mathematically-oriented toolkits and biophysically-oriented toolkits in a deliberately pedagogical manner.

This book is for researchers, engineers and students who wish to use the Brain Dynamics Toolbox to construct and explore their own dynamical models. It assumes a working knowledge of MATLAB and some familiarity with the basic numerical methods for solving initial value problems in nonlinear dynamical systems (e.g. ode45). Advance object-oriented programming techniques are not required. A new model can typically be written in less than 100 lines of standard MATLAB code. Doing so primarily involves (i) defining the differential equations for the solver and (ii) defining the names and sizes of the model's parameters for the graphical user interface. Once that has been done, the model can be run interactively and visualised in any number of ways with no additional programming effort.

As of this version (2018a) the toolbox supports approximately one dozen solver routines and a similar number of visualization tools. That list continues to grow with each new version of the toolbox. Its ongoing development has benefited greatly from the enthusiastic support of the Systems Neuroscience Group at QIMR Berghofer Medical Research Institute. In particular, we gratefully acknowledge Matthew Aburn's contribution to the design of the SDE solvers. We also thank James Roberts for his comments on the original manuscript.

Australia
March 2018

Stewart Heitmann
Michael Breakspear

Contents

Chapter 1
Introduction

The *Brain Dynamics Toolbox* is an open-source toolbox for simulating and exploring non-linear dynamical systems in MATLAB. It includes a graphical user interface for exploring the dynamics interactively (Figure 1.1) as well as command-line tools for scripting large-scale simulations. The toolbox is intended for researchers and students who wish to use dynamical systems to investigate the theoretical basis of brain function. As such, it supports the three major classes of dynamical systems that typically arise in computational neuroscience — Ordinary Differential Equations (ODEs), Delay Differential Equations (DDEs) and Stochastic Differential Equations (SDEs). Nonetheless the toolbox is ostensibly a platform for solving initial value problems and can be readily applied to dynamical systems from any problem domain. Its major benefit is that it brings differential equations to life as interactive simulations that can be semi-automated with workspace commands or fully-automated with custom scripts [17].

The hub-and-spoke architecture of the toolbox (Figure 1.2) allows unlimited combinations of solvers and plotting tools to be applied to any model with no additional coding effort. The combinatorial power of this design frees the researcher from the burden of coding bespoke graphical interfaces and solver scripts for each new model that they construct. The architecture is deliberately modular too. It allows researchers to augment the toolbox with their own custom display panels or solver routines. New modules can be loaded a run-time without having to modify the toolbox source code. The toolbox currently ships with display modules (panels) for visualising mathematical equations, time plots, phase portraits, space-time plots, bifurcation diagrams, Hilbert phases, linear correlations and model-specific auxiliary functions, among others. That list continues to grow. It also makes extensive use of the standard ODE and DDE solvers that are shipped with MATLAB as well as providing several new ones — notably the SDE solvers (`sdeEM`, `sdeSH`) and the fixed-step Euler method (`odeEul`) for ODEs.

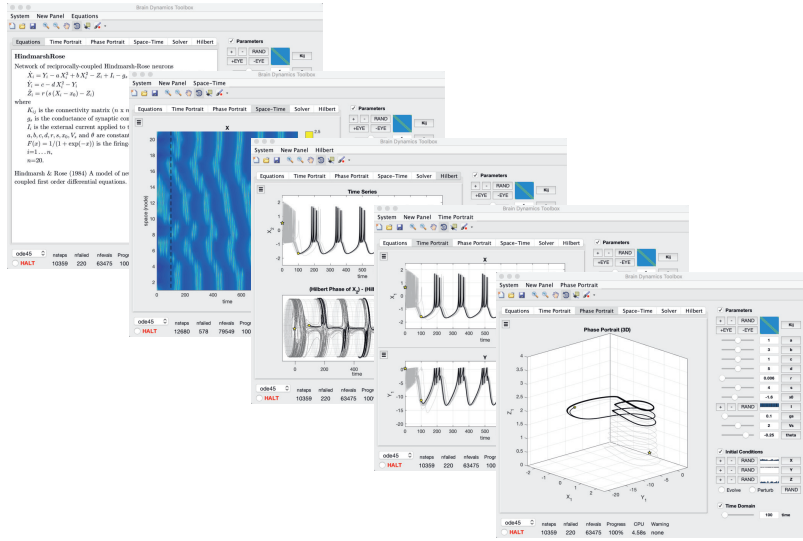

Fig. 1.1 Screenshots of the graphical user interface showing a selection of display panels from the same model — in this case a ring of n=20 Hindmarsh-Rose neurons. Left to right: (i) Mathematical equations rendered with LaTeX. (ii) Space-time plot of the neural activity on the ring. (iii) Hilbert phases of each neuron's membrane potential. (iv) Time plots. (v) Phase portrait of the three state variables for a selected neuron. In all cases, the model's parameters and initial conditions appear on the right-hand side of the screenshot. All neurons receive identical stimulation (`I=1.5*ones(n,1)`).

Overall, the toolbox uses the same approach for solving differential equations as the existing MATLAB solvers (e.g. `ode45`). Ostensibly, the user defines the right-hand side of their dynamical system,

$$\frac{d\mathbf{Y}}{dt} = F(t, \mathbf{Y}, p),$$

as a MATLAB function of the form `dYdt=F(t,Y,p)` where `Y` is a vector of state variables and `p` contains parameter constants. A handle to that function is then passed to the solver which calls it repeatedly to integrate the equations forward in time from a given set of initial conditions.

The Brain Dynamics Toolbox automates the process of calling the solver and plotting the output on the user's behalf. To do so, it requires additional information about the system's state variables and parameters. Those details and others are encapsulated in a specially formatted data structure which we call the *system structure*. Listing 1.1 shows the system structure for the Hindmarsh-Rose model from Figure 1.1. It is nothing more than a MATLAB structure whose fields follow toolbox conventions. The exact procedures for creating a system structure for a new model are described in Chapters 2-4. In

general, the process involves writing a small script that populates the system structure with a handle to the user-defined function and configures the names and initial values of the state variables and parameters. That system structure is then loaded into the graphical user interface which runs the model. A collection of example scripts that demonstrate key aspects of model building are provided with the toolbox (Table 1.1). A typical model has fewer than 100 lines of standard MATLAB code. Advanced object-oriented programming techniques are not required.

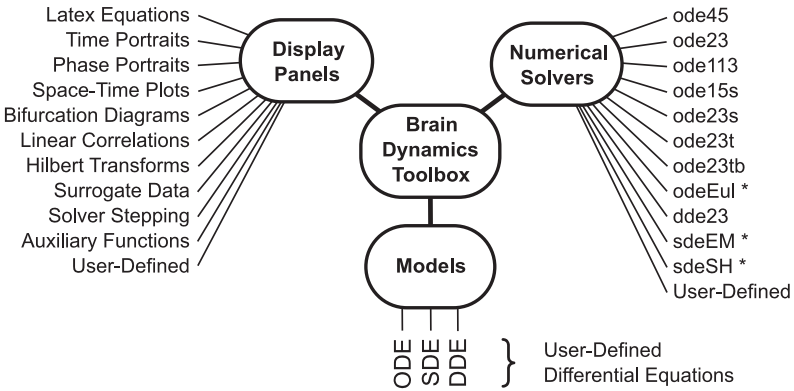

Fig. 1.2 Hub-and-spoke architecture of the Brain Dynamics Toolbox. The toolbox acts as a central hub that connects user-defined models with inter-changeable solver routines and display panels. Solvers marked by an asterisk are unique to the toolbox.

Listing 1.1 The system structure for the Hindmarsh-Rose model shown in Figure 1.1.

```
sys =
struct with fields:
     odefun: @odefun
     pardef: [12x1 struct]
     vardef: [3x1 struct]
      tspan: [0 1000]
  odesolver: {@ode45  @ode23  @ode113  @odeEul}
  odeoption: [1x1 struct]
     panels: [1x1 struct]
```

Table 1.1 Example models supplied with the toolbox.

Model	Type	Description
`BrownianMotion`	SDE	Geometric Brownian motion.
`BTF2003/SDE/DDE`	all	Neural masses with noise and delays [5, 18].
`DFCL2009`	ODE	Point neural-mass model of chaotic EEG [8].
`FitzhughNagumo`	ODE	Fitzhugh-Nagumo neural oscillator [13, 36].
`FRRB2012`	SDE	Multistable neural oscillators with noise [14].
`HindmarshRose`	ODE	Network of Hindmarsh-Rose neurons [20].
`HopfieldNet`	ODE	Hopfield associative memory network [22].
`KloedenPlaten446`	SDE	Ito equation (4.46) in Kloeden & Platen [24].
`KuramotoNet`	ODE	Network of Kuramoto oscillators [3, 19, 27].
`LinearODE`	ODE	Linear ODE in two variables.
`OrnsteinUhlenbeck`	SDE	Ornstein-Uhlenbeck noise processes.
`RFB2017`	SDE	Neural-mass with multiplicative noise [33].
`SwiftHohenberg1D`	PDE	Swift-Hohenberg Equation in 1D space. †
`VanDerPolOscillators`	ODE	Network of Van der Pol oscillators [41, 32].
`WaveEquation1D`	PDE	Wave Equation in one spatial dimension. †
`WilleBakerEx3`	DDE	Example 3 from Willé and Baker [44].

† The Partial Differential Equations (PDEs) in this table have been transformed into ODEs by discretising space using the method of lines [38].

1.1 Download and dependencies

The source code for the toolbox is distributed freely under the BSD 2-clause license. It can be downloaded from the Brain Dynamics Toolbox website.

```
http://www.bdtoolbox.org
```

The toolbox requires MATLAB R2014b or newer. It will not work with Octave. The current version of the toolbox (2018a) does not depend on any additional MATLAB toolbox although previous versions did. The version number of the toolbox need not match the version number of MATLAB itself.

1.2 Installation

Installation simply involves unzipping the source code into a location of your choosing. The main scripts are located in the top level of the *bdtoolkit* directory. That directory must be in the MATLAB search path. It is also advisable to include the *bdtoolkit/models* directory in the search path if you intend to run any of the example models therein.

```
>> addpath bdtoolkit-2018a
>> addpath bdtoolkit-2018a/models
```

The display panels and solver routines are installed in the *bdtoolkit/panels* and *bdtoolkit/solvers* directories respectively. The toolbox automatically adds those directories to the search path whenever it is run. User-defined models, panels and solvers need not be located in the installation directory but they must be reachable on the MATLAB search path. Multiple versions of the toolkit can co-exist on the same machine provided that only one version is ever in the search path at any time. The toolbox can be un-installed by simply deleting the *bdtoolkit* directory and all that it contains.

1.3 Getting started

Every instance of a model is defined by a system structure (`sys`) that contains the function handles and parameter definitions needed to run it. New system structures are typically constructed using a model-specific function for that very purpose. Whereas an existing system structure can be loaded from a *mat* file. Either way, the model is run by loading its system structure into the graphical user interface (`bdGUI`) .

The following example shows how to load the `sys` structure for the Hindmarsh-Rose [20] model from the *HindmarshRose.mat* file.

```
>> load HindmarshRose.mat sys
>> bdGUI(sys);
```

This particular *mat* file can be found in the *bdtoolkit/models* directory. The functions that accompany it are located in the *HindmarshRose.m* file in the same directory. The model will fail to load if those functions are not reachable on the MATLAB search path.

The `bdGUI` command will automatically prompt the user to load a model from a *mat* file if it is called with no input parameter. Once it loads, the solutions is automatically computed based on the current parameters in the control panel (Figure 1.3). Furthermore, that solution is automatically updated whenever those controls are adjusted. The solver can be suspended by activating the red HALT button in the bottom-left corner of the user interface.

The graphical controls can represent scalar, vector or matrix values depending on how the model was defined. Scalar values appear as slider controls whereas vectors and matrices have push-buttons for manipulating their contents. Additional editing options can be invoked by clicking the buttons that are labelled with the names of the system parameters. The time slider at the bottom of the control panel is used to trim unwanted transients from the output. Almost all display panels have menu options that allow the transient time window to be hidden from view.

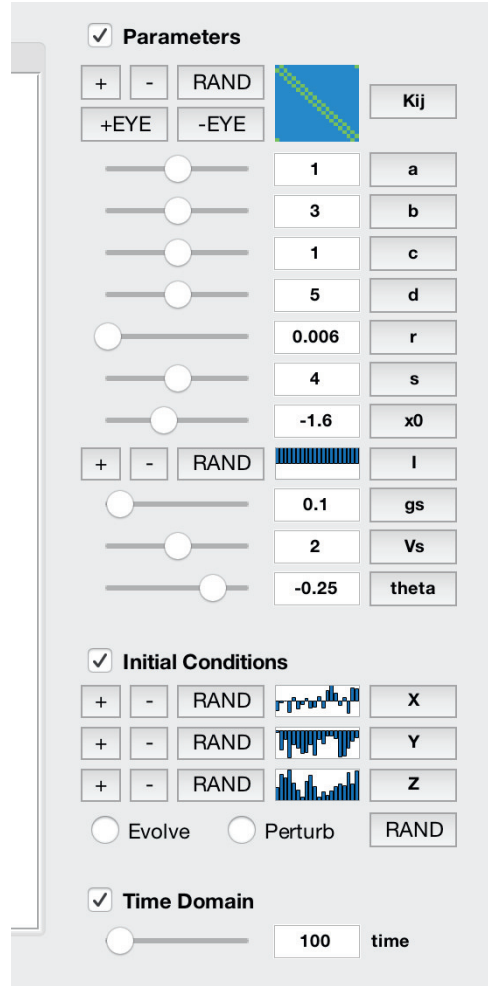

Fig. 1.3 Control panel for the Hindmarsh-Rose model. The controls are arranged in rows with the model's parameters (`Kij`, `a`, `b`, `c`, `d`, `r`, `s`, `x0`, `I`, `gs`, `Vs`, `theta`) grouped separately from its state variables (`X`, `Y`, `Z`). In this case, `Kij` is a 20×20 connectivity matrix. The parameter I and the state variables X, Y, Z are each 20×1 vectors. The values of the state variables correspond to the initial conditions of the system. The remaining controls are all scalars. Each one has a slider and an edit box. Vector controls have three push-buttons (`+`, `-`, `RAND`) for gross manipulations. Matrices have two additional push-buttons (`+EYE`, `-EYE`) for manipulating the diagonal. All controls have extra editing options which can be accessed by clicking the button that displays its name. Refer to the tool-tips for the meaning of specific buttons.

Evolving the initial conditions.

The initial conditions for each simulation run can be automatically replaced with the final state of the previous run by activating the *Evolve* button on the control panel. This is useful for continuing a solution in parameter space or for breaking long simulations into smaller consecutive runs.

Perturbing the initial conditions.

The initial conditions can also be automatically perturbed prior to each simulation run by activating the *Perturb* button. This is useful for exploring the

sensitivity of the system to initial conditions or for escaping a solution that has just become unstable. The perturbation is uniformly random with a maximum span that is equivalent to 5% of the axes limits of each variable. It is applied to the simulation without replacement, meaning that it does not alter the value of the initial conditions in the control panel.

Constructing new system structures.

Some aspects of any model are inevitably fixed at construction. In the previous example, the number of neurons in the Hindmarsh-Rose model was fixed at n=20. The *HindmarshRose* function can be used to construct system structures with other configurations.

```
function sys = HindmarshRose(Kij)
```

It takes an $n \times n$ connectivity matrix as input (`Kij`) as input and returns a system structure (`sys`) for a network of n neurons. The following example constructs a system of n=21 randomly connected Hindmarsh-Rose neurons and loads it into the graphical user interface.

```
>> n = 21;
>> Kij = rand(n);
>> sys = HindmarshRose(Kij);
>> bdGUI(sys);
```

We can similarly construct a network of n=47 neurons using a physiological connectivity matrix from the CoCoMac database [25].

```
>> load cocomac047.mat CIJ
>> sys = HindmarshRose(CIJ);
>> bdGUI(sys);
```

General approach to model configuration.

The majority of models shipped with the toolbox follow the same approach as outlined above. The `sys` structure is generated by a model-specific script that determines the size of the model (the number of dynamical equations) from one of the input parameters (usually a connectivity matrix). However each script is unique so use the *help* command to ascertain the details of each model.

```
>> help HindmarshRose
```

1.4 The display panels

Display panels are plug-in modules that can be loaded and unloaded from the graphical user interface at run-time. Each one provides a different visualisation of the computed solution. The toolbox ships with a collection of display panel classes in the *bdtoolkit/panels* directory (Table 1.2). All panels in that directory are automatically included in the *New Panel* toolbar menu.

The *Time Portrait* panel (Figure 1.4) is the principal display panel for inspecting time-series data. It is actually two panels in one since the upper and lower sub-panels operate independently. Each one shows the time course of a state variable which is selected from the pull-down menu icon next to the axes. If the selected time-series is part of a vector or matrix of state variables then its other elements are plotted in the background as light-grey traces.

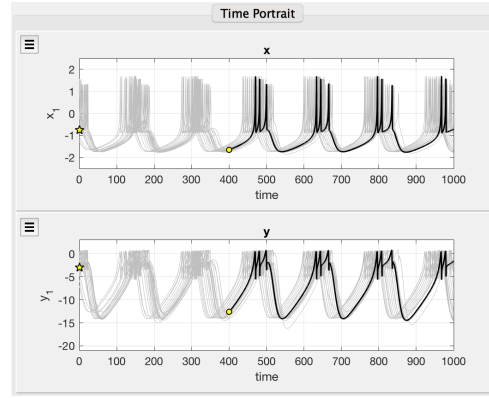

Fig. 1.4 Time Portrait panel.

Table 1.2 The display panel classes.

Panel Class	Menu Name	Description
`bdAuxiliary`	Auxiliary	Model-specific plotting functions.
`bdBifurcation`	Bifurcation	Bifurcation diagrams of state variables.
`bdCorrPanel`	Correlation	Linear correlation for multi-variate data.
`bdHilbert`	Hilbert	Hilbert phase of a selected state variable.
`bdLatexPanel`	Equations	Mathematical equations using LaTeX.
`bdPhasePortrait`	Phase Portrait	Phase portraits of selected state variables.
`bdSolverPanel`	Solver	Solver step-size and error tolerances.
`bdSpaceTime`	Space-Time	Space-time plots for multi-variate data.
`bdSurrogate`	Surrogate	Phase randomised surrogate data.
`bdTimePortrait`	Time Portrait	Time plots for two selected state variables.
`bdTrapPanel`	Trap	Tool for debugging display panels.

The transient part of the trajectory — as determined by the position of the time slider in the control panel — is usually greyed out. This is a common theme among all display panels. By convention, the initial conditions are marked by a yellow pentagram and the start of the non-transient time window is marked by a yellow circle. The transient part can be hidden altogether by toggling the *Transients* menu option (Figure 1.5). The yellow markers can likewise be hidden by toggling the *Markers* menu option.

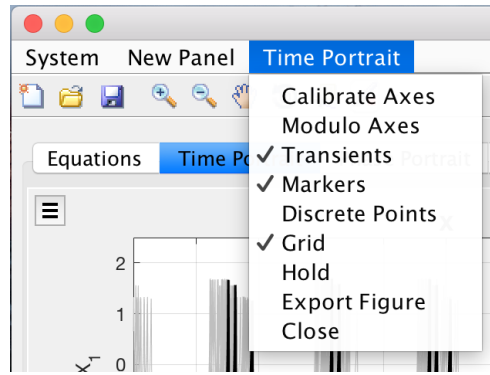

Fig. 1.5 Time Portrait menu.

The toolbar menu is context sensitive. It displays only those menus that relate to the currently active display panel. Again, there are common themes. The majority of panels have a *Calibrate Axes* menu item. It determines the lower and upper limits of the axes that best fit the plotted data and adjusts the limits of the relevant controls in the control panel. The new limits apply globally to all display panels. Calibrating the axes is such a common operation that it soon becomes second nature.

The *Grid* and *Hold* menu options are supported by most panels. The former toggles the axes grid lines (`grid on|off`). The latter toggles the drawing mode (`hold on|off`) so that successive simulation plots can be superimposed on the panel axes. The *Discrete Points* menu option is supported by many panels but not all. It causes the panel to plot the trajectory as discrete time points rather than as a continuous line. Those points corresponds to the exact time steps taken by the solver.

The *Modulo Axes* menu option is only supported by a few display panels, notably the *Time Portrait* and the *Phase Portrait*. It instructs those panels to wrap the plot lines at the boundaries rather than clipping them. It is useful for plotting periodic data such as phase angles.

The *Export Figure* menu item exports a copy of the panel's axes to a new figure that is independent of `bdGUI`. This feature is useful for preparing figures for publication. The *Close* menu item closes the currently active panel and removes its menu from the toolbar.

1.5 The workspace interface

The `bdGUI` command returns a handle to the internal states of the graphical user interface that it creates.

```
>> gui = bdGUI(sys);
```

The public properties of that class handle (Table 1.3) provide a direct interface between the user's private workspace and the workspace of the model running in the graphical user interface. Hence it is possible to interactively analyse the model's output using third party tools such as the Brain Connectivity Toolbox [34]. Listing 1.2 illustrates the basic approach. It uses the `gui` handle to read the parameters and state variables of the running model and to plot the computed solution in a separate figure (Figure 1.6). The scope and lifespan of the data in that figure belongs exclusively to the user's workspace. That data could be passed to any third-party application in the same way.

It is even possible to manipulate the model's parameters and initial conditions as it runs. It is worth emphasising that the `gui` object is a *handle* to the internal states of the graphical user interface rather than a separate copy. So any changes to its properties are instantly reflected in the graphical user interface and vice versa. The parameters of the model can be manipulated by assigning values directly to the relevant fields of the `par` structure. Likewise, the initial conditions can be manipulated by assigning values to the fields of the `var0` structure. Notice the distinction between the initial conditions (`var0`) and the solution variables (`var`). The former is read-write whereas the latter is read-only.

Table 1.3 Public properties of the `bdGUI` class.

Property	Type	Description	Access
`version`	`string`	Toolbox version string '2018a'.	read-only
`fig`	`figure`	Graphics handle to the application figure.	read-write
`par`	`struct`	Parameters of the model (by name).	read-write
`var0`	`struct`	Initial conditions (by name).	read-write
`var`	`struct`	Solution variables (by name).	read-only
`t`	`double`	Time steps taken by the solver.	read-only
`tindx`	`logical`	Indexes the non-transient time steps.	read-only
`lag`	`struct`	DDE lag parameters (by name).	read-write
`sys`	`struct`	Current state of the model's `sys` structure.	read-only
`sol`	`struct`	Current solution as returned by the solver.	read-only
`panels`	`struct`	Properties of the individual display panels.	read-only

Listing 1.2 Using the public properties of the `bdGUI` class to access the current state of parameters and variables in the Hindmarsh-Rose model. The output of the plot command (line 27) is shown in Figure 1.6.

```
1  >> load HindmarshRose.mat sys
2  >> gui = bdGUI(sys);
3  >> gui.par
4  ans =
5    struct with fields:
6        Kij: [20x20 double]
7          a: 1
8          b: 3
9          c: 1
10         d: 5
11         r: 0.0060
12         s: 4
13        x0: -1.6000
14         I: [20x1 double]
15        gs: 0.1000
16        Vs: 2
17     theta: -0.2500
18
19 >> gui.var
20 ans =
21   struct with fields:
22     x: [20x10579 double]
23     y: [20x10579 double]
24     z: [20x10579 double]
25
26 >> figure;
27 >> plot(gui.t,gui.var.x);
```

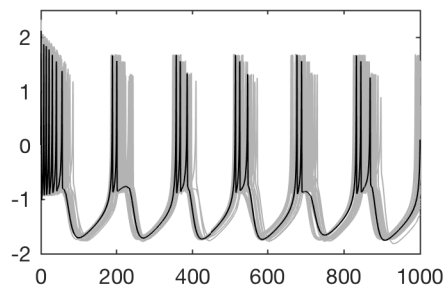

Fig. 1.6 The output produced by Listing 1.2.

Workspace scripting.

The workspace interface is particularly convenient for scripting quick parameter surveys. The following brief script illustrates how to ramp the current (`I`) applied to all neurons in the Hindmarsh-Rose model. In this case the `gui.par.I` parameter is a vector of n values (one per neuron) which together are ramped from 0 to 2 in increments of 0.1.

```
n = 21;
Kij = rand(n);
sys = HindmarshRose(Kij);
gui = bdGUI(sys);
for I=0:0.1:2
   gui.par.I = I*ones(n,1);
end
```

The graphical interface automatically recomputes the solution each time the `gui.par.I` parameter is assigned a new value. MATLAB will wait for that computation to complete before performing the next instruction in the script. Hence there is no need to insert a *wait* command in the for-loop.

The *Evolve* button can be used during a parameter survey to follow a solution far from its initial conditions. Any transient dynamics that arise with each step change in the parameter can be hidden using the time domain slider. An entire branch of steady-state solutions can thus be plotted neatly in the bifurcation panel as the survey progresses.

1.6 The command-line tools

The toolbox includes a suite command-line tools (Chapter 7) for scripting large-scale simulations without having to invoke the graphical user interface. These tools provide the same level of control as the workspace interface but are intended for large parameter surveys that run in batch mode. The primary command-line tool is `bdSolve` which solves an initial-value problem for a given system structure. The `bdEval` command interpolates the computed solution for a given set of time points. It is analogous to the MATLAB `deval` command except that it also works for solutions returned by the toolbox's custom solver routines. The toolbox also has some useful utility functions such as `bdGetValue` and `bdSetValue` which allow the parameters and initial conditions in the `sys` structure to be accessed by name. Another important tool is `bdSysCheck` which validates the format of a user-defined system structure during the development cycle.

1.7 A worked example

We complete this introductory chapter by simulating traveling waves in a ring of Hindmarsh-Rose [20] neurons using the example model shipped with the toolbox. It is a model of spike-bursting behaviour where each neuron is represented by its membrane potential $X_i(t)$, a fast recovery variable $Y_i(t)$ and a slow recovery variable $Z_i(t)$.

$$\begin{aligned}
\dot{X}_i &= Y_i - aX_i^3 + bX_i^2 - Z_i + I_i - g_s(X_i - V_s)\sum K_{ij}F(X_j - \theta) \\
\dot{Y}_i &= c - dX_i^2 - Y_i \\
\dot{Z}_i &= r(s(X_i - x_0) - Z_i
\end{aligned}$$

The two recovery variables represent ionic currents in the neural membrane. The fast current contributes to the shape of each spike and the slow current contributes to the envelope of the spike bursting. The connectivity matrix K_{ij} defines the topology of the network and the synaptic conductance g_s governs the strength of the coupling between neurons. The parameter I_i represents an external stimulation current. It is applied to each neuron independently. The other parameters are not important here, so we omit them for brevity.

We begin by constructing an instance of the Hindmarsh-Rose model using a connectivity matrix (`Kij`) that defines a ring network.

```
>> n = 21;
>> Kij = circshift(eye(n),1) ...
       + circshift(eye(n),-1);
>> sys = HindmarshRose(Kij);
```

The `sys` structure is loaded into the graphical interface in the usual manner.

```
>> gui = bdGUI(sys);
```

We wish to drive only one neuron in the ring and observe its influence on its neighbouring neurons. For demonstration purposes, we use the `gui` handle rather than the graphical control panel to apply a non-zero current to the 11^{th} neuron in the ring.

```
>> gui.par.I(11) = 2;
```

That level of current induces repetitive bursts of spikes in the membrane potential of the stimulated neuron (X_{11} in Figure 1.7a) but those spikes fail to propagate to the neighbouring neurons in the ring (as shown by the space-time panel in Figure 1.7b) because the synaptic conductance is too weak (g_s=0.1). Using the slider control, we gradually increase the synaptic conductance and observe the emergence of spreading activity (Figure 1.7c) at g_s=0.37. That spreading activity attenuates rapidly with distance. It is not

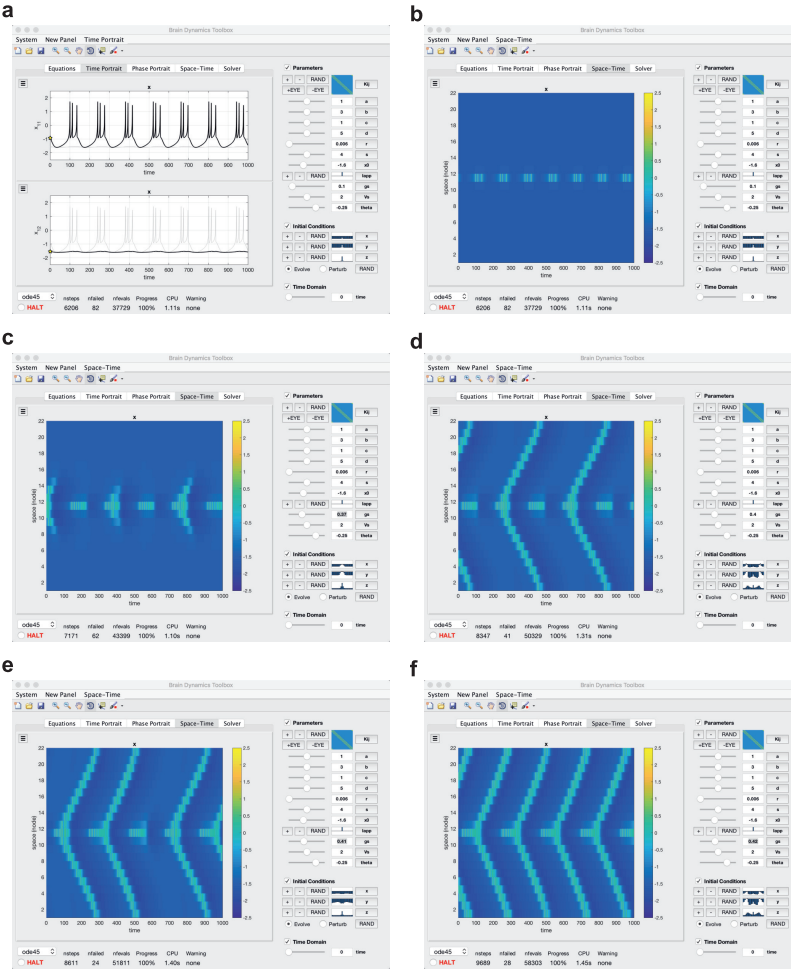

Fig. 1.7 Propagating waves in a ring of n=20 Hindmarsh-Rose neurons. (**a**) Time portraits for the stimulated neuron (X_{11}) in the upper panel and its neighbouring neuron (X_{12}) in the lower panel. The synaptic conductance is g_s=0.1. (**b**) Space-time plot of $X_i(t)$ for all neurons in the ring when g_s=0.1. The horizontal axis is time. The vertical axis represents the spatial position of each neuron. It has periodic boundary conditions in the case. The colour scale indicates the amplitude of $X(t)$. (**c-f**) Space-time plots for the same model with g_s=0.37, g_s=0.40, g_s=0.41 and g_s=0.42 respectively.

until the synaptic conductance is increased to g_s=0.4 that the model supports self-sustaining propagating waves (Figure 1.7d). They travel around the ring in opposite directions until they collide and annihilate one another. Even then those waves are only emitted from the stimulation site on every second burst cycle. Increasing the synaptic conductance of the network further still, we observe waves emitted every 2:3 cycles for g_s=0.41 (Figure 1.7e) and on every cycle for g_s=0.42 (Figure 1.7f). Notice how we eliminated unwanted transients from our simulations by using the *Evolve* button to gradually follow the solution from g_s=0.1 to g_s=0.42.

This brief exercise illustrates how the toolbox can be used to quickly simulate and explore complex dynamical phenomenon. The remaining chapters of this book describe how to use the toolbox to construct and explore your own models.

Handbook for the **Brain Dynamics Toolbox**

Stewart Heitmann
Michael Breakspear

www.bdtoolbox.org

amazon.com

References

1. Aburn M.J. (2017) Critical fluctuations and coupling of stochastic neural mass models. Ph.D. thesis, University of Queensland.
2. Bower J.M., Beeman D. (1998) The book of Genesis: exploring realistic neural models with the General Neural Simulation System. Telos, Springer, New York.
3. Breakspear M., Heitmann S., Daffertshofer A. (2010) Generative models of cortical oscillations: Neurobiological implications of the Kuramoto model. Frontiers in Human Neuroscience 4, 190.
4. Breakspear M., Terry J. (2002) Detection and description of non-linear interdependence in normal multichannel human EEG data. Clinical Neurophysiology 113(5), 753.
5. Breakspear M., Terry J.R., Friston K.J. (2003) Modulation of excitatory synaptic coupling facilitates synchronization and complex dynamics in a biophysical model of neuronal dynamics. Network: Computation in Neural Systems 14(4), 703–732.
6. Carnevale N.T., Hines M.L. (2006) The NEURON book. Cambridge University Press.
7. Clewley R. (2012) Hybrid Models and Biological Model Reduction with PyDSTool. PLOS Computational Biology 8(8), e1002628. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1002628.
8. Dafilis M.P., Frascoli F., Cadusch P.J., Liley D.T. (2009) Chaos and generalised multistability in a mesoscopic model of the electroencephalogram. Physica D: Nonlinear Phenomena 238(13), 1056–1060. ISSN 01672789. doi:10.1016/j.physd.2009.03.003.
9. Dankowicz H., Schilder F. (2013) Recipes for Continuation. SIAM. ISBN 978-1-61197-256-6.
10. Dhooge A., Govaerts W., Kuznetsov Y.A. (2003) MATCONT: a MATLAB package for numerical bifurcation analysis of ODEs. ACM Transactions on Mathematical Software (TOMS) 29(2), 141–164.
11. Doedel E.J., Champneys A.R., Fairgrieve T.F., Kuznetsov Y.A., Sandstede B., Wang X. (1998). AUTO 97: Continuation and bifurcation software for ordinary differential equations (with HomCont).
12. Ermentrout B. (2002) Simulating, analyzing, and animating dynamical systems: a guide to XPPAUT for researchers and students. SIAM. ISBN 978-0-89871-506-4.
13. FitzHugh R. (1955) Mathematical models of threshold phenomena in the nerve membrane. Bulletin of Mathematical Biology 17(4), 257–278.
14. Freyer F., Roberts J.A., Ritter P., Breakspear M. (2012) A canonical model of multistability and scale-invariance in biological systems. PLOS Computational Biology 8(8), e1002634.

15. Gardiner C. (2009) Stochastic Methods: A Handbook for the Natural and Social Sciences. Springer, 4th edition. ISBN 978-3-540-70712-7.
16. Goodman D.F.M., Brette R. (2013) BRIAN simulator. Scholarpedia 8(1), 10883. ISSN 1941-6016. doi:10.4249/scholarpedia.10883.
17. Heitmann S., Aburn M.J., Breakspear M. (2017) The Brain Dynamics Toolbox for Matlab. bioRxiv doi:10.1101/219329.
18. Heitmann S., Breakspear M. (2017) Putting the "dynamic" back into dynamic functional connectivity. bioRxiv doi:10.1101/181313.
19. Heitmann S., Ermentrout G.B. (2015) Synchrony, waves and ripple in spatially coupled Kuramoto oscillators with Mexican hat connectivity. Biological Cybernetics 109(3), 333–347. ISSN 0340-1200, 1432-0770. doi:10.1007/s00422-015-0646-6.
20. Hindmarsh J.L., Rose R.M. (1984) A model of neuronal bursting using three coupled first order differential equations. Proceedings of the Royal Society of London B: Biological Sciences 221(1222), 87–102.
21. Hodgkin A.L., Huxley A.F. (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. The Journal of Physiology 117(4), 500–544. ISSN 0022-3751.
22. Hopfield J.J. (1982) Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences 79(8), 2554–2558.
23. Jacobs K. (2010) Stochastic processes for physicists: understanding noisy systems. Cambridge University Press.
24. Kloeden P.E., Platen E. (1992) Numerical solution of stochastic differential equations springer-verlag. Number 23 in Applications of Mathematics. Springer-Verlag. ISBN 3-540-54062-8.
25. Kotter R. (2004) Online retrieval, processing, and visualization of primate connectivity data from the CoCoMac database. Neuroinformatics 2(2), 127–144.
26. Kottwitz S. (2011) LaTeX beginner's guide. Packt Publishing Ltd.
27. Kuramoto Y. (1984) Chemical oscillations, waves, and turbulence. Dover Publications, Mineola, New York.
28. Lamport L. (1994) LaTeX: A Document Preparation System, 2nd Edition. Addison-Wesley Professional., 2nd edition. ISBN 978-0-201-52983-8.
29. Marple L. (1999) Computing the discrete-time analytic signal via FFT. IEEE Transactions on Signal Processing 47(9), 2600–2603. ISSN 1053-587X. doi: 10.1109/78.782222.
30. Maruyama G. (1955) Continuous Markov processes and stochastic equations. Rendiconti del Circolo Matematico di Palermo 4(1), 48–90.
31. Park Y., Heitmann S., Ermentrout B. (2017) The utility of phase models in studying neuronal synchronization. In Computational models of brain and behavior, pages 493–504. John Wiley & Sons, 1st edition. ISBN 978-1-119-15906-3. Ed Ahmed A Moustafa.
32. Rand R.H., Holmes P.J. (1980) Bifurcation of periodic motions in two weakly coupled van der Pol oscillators. International Journal of Non-Linear Mechanics 15(4-5), 387–399.
33. Roberts J.A., Friston K.J., Breakspear M. (2017) Clinical Applications of Stochastic Dynamic Models of the Brain, Part I: A Primer. Biological Psychiatry: Cognitive Neuroscience and Neuroimaging ISSN 24519022. doi:10.1016/j.bpsc.2017.01.010.
34. Rubinov M., Sporns O. (2010) Complex network measures of brain connectivity: Uses and interpretations. NeuroImage 52(3), 1059–1069. ISSN 10538119. doi: 10.1016/j.neuroimage.2009.10.003.
35. Ruemelin W. (1982) Numerical treatment of stochastic differential equations. SIAM Journal on Numerical Analysis 19(3), 604–613.

36. Sanz-Leon P., Knock S.A., Spiegler A., Jirsa V.K. (2015) Mathematical framework for large-scale brain network modeling in The Virtual Brain. Neuroimage 111, 385–430.
37. Sanz Leon P., Knock S.A., Woodman M.M., Domide L., Mersmann J., McIntosh A.R., Jirsa V. (2013) The Virtual Brain: a simulator of primate brain network dynamics. Frontiers in Neuroinformatics 7. ISSN 1662-5196. doi:10.3389/fninf.2013.00010.
38. Shampine L.F., Gladwell I., Thompson S. (2003) Solving ODEs with Matlab. Cambridge University Press.
39. Smythe J., Moss F., McClintock P.V., Clarkson D. (1983) Ito versus Stratonovich revisited. Physics Letters A 97(3), 95–98.
40. The Mathworks (2017). MATLAB & Simulink: Text with mathematical expression using LaTeX.
41. Van der Pol B. (1934) The nonlinear theory of electric oscillations. Proceedings of the Institute of Radio Engineers 22(9), 1051–1086.
42. Van Kampen N.G. (1981) Ito versus Stratonovich. Journal of Statistical Physics 24(1), 175–187.
43. Van Kampen N.G. (1992) Stochastic processes in physics and chemistry, volume 1. Elsevier.
44. Wille D.R., Baker C.T. (1992) DELSOL - a numerical code for the solution of systems of delay-differential equations. Applied Numerical Mathematics 9(3-5), 223–234.

Index

The official guide to the **Brain Dynamics Toolbox**

The *Brain Dynamics Toolbox* is an open-source software toolbox for simulating dynamical systems in neuroscience. It is for students and researchers who wish to explore numerical models of brain function using Matlab. It includes a graphical platform that brings differential equations to life as interactive simulations, and includes command-line tools for scripting large-scale simulations in batch mode.

The toolbox provides a convenient platform for solving initial-value problems in non-linear dynamical systems. It supports the three major classes of differential equations in Computational Neuroscience — Ordinary Differential Equations (ODEs), Delay Differential Equations (DDEs) and Stochastic Differential Equations (SDEs). It can readily be applied to differential equations from other problem domains too.

Dr Stewart Heitmann is a Senior Software Engineer & Research Associate at QIMR Berghofer Medical Research Institute. He has a PhD in Computational Neuroscience from UNSW and post-doctoral training in Mathematical Neuroscience from the University of Pittsburgh. He studies pattern formation in neural systems.

Professor Michael Breakspear is Senior Scientist and Group Leader of the Systems Neuroscience Group at QIMR Berghofer Medical Research Institute. He is interested in the fundamental principles of large-scale brain dynamics, how these arise from cortical architectures, and how they underpin cognitive operations.