The Brain Dynamics Toolbox for Matlab

Stewart Heitmann, Matthew Aburn, Michael Breakspear

QIMR Berghofer Medical Research Institute
300 Herston Road, Herston QLD 4006, Australia

Abstract

The Brain Dynamics Toolbox provides a graphical tool for simulating user-defined dynamical systems in MATLAB. It supports three classes of differential equations commonly used in computational neuroscience: Ordinary Differential Equations, Delay Differential Equations and Stochastic Differential Equations. The design of the graphical interface fosters intuitive exploration of the dynamics, yet there is no barrier to scripting large-scale simulations and parameter explorations. System variables and parameters may range in size from simple scalars to large vectors or matrices. The toolbox is intended for dynamical models in computational neuroscience but it can be applied to continuous dynamical systems from any domain.

Keywords: initial-value problems, differential equations, numerical integration, visualization, brain dynamics

1. Introduction

Computational neuroscience relies heavily on numerical methods for simulating non-linear models of brain dynamics. Software toolkits are the manifestation of those endeavors. Each one represents an attempt to balance mathematical flexibility with computational convenience. Toolkits such as GENESIS [1], NEURON [2] and BRIAN [3] provide convenient methods to simulate conductance-based models of single neurons and networks thereof. The Virtual Brain [4] scales up that approach to the macroscopic dynamics of the whole brain. It combines neural field models [5] with anatomical connectivity datasets for the purpose of generating realistic EEG, MEG and fMRI data. Toolkits such as AUTO [6], XPPAUT [7], MATCONT [8], PyDSTool [9] and CoCo [10] provide numerical methods from applied mathematics for

analyzing non-linear dynamical systems. These toolkits are highly applicable to computational neuroscience but assume a substantial background in mathematical theory. Many of the toolkits described above also require substantial computer programming effort from the user.

2. Problems and Background

In our experience of publishing and teaching computational neuroscience, the existing toolkits often present technical barriers to a broader audience. The Brain Dynamics Toolbox aims to bridge that gap by allowing those with diverse backgrounds to explore neuronal dynamics through phase space analysis, time series exploration and other methods with minimal programming burden. A custom model can typically be implemented with our toolbox in fewer than 100 lines of standard MATLAB code. Object-oriented programming techniques are not required. Once the model is implemented, it can be loaded into the graphical interface (Figure 1) for interactive simulation or run without the graphical interface using command-line tools. The graphical controls are themselves accessible to the user as workspace variables. This makes it easy to use the MATLAB command window to orchestrate quick parameter surveys within the graphical interface. The command-line tools are useful for scripting larger surveys since they utilize the same toolbox infrastructure but do not invoke the graphical interface. The Brain Dynamics Toolbox thus provides intuitive access to neurodynamical modeling tools while retaining the ability to automate large-scale simulations.

3. Software Framework

The toolbox operates on user-defined systems of Ordinary Differential Equations (ODEs), Delay Differential Equations (DDEs) and Stochastic Differential Equations (SDEs). The details differ slightly for each type but the overall approach is the same. The right-hand side of the dynamical system,

$$\frac{dY}{dt} = F(t, Y),$$

is implemented as a matlab function of the form `dYdt=F(t,Y)`. The toolbox takes a handle to that function and passes it to the relevant solver routine on the user's behalf. The solver repeatedly calls `F(t,Y)` in the process of computing the evolution of $Y(t)$ from a given set of initial conditions. The

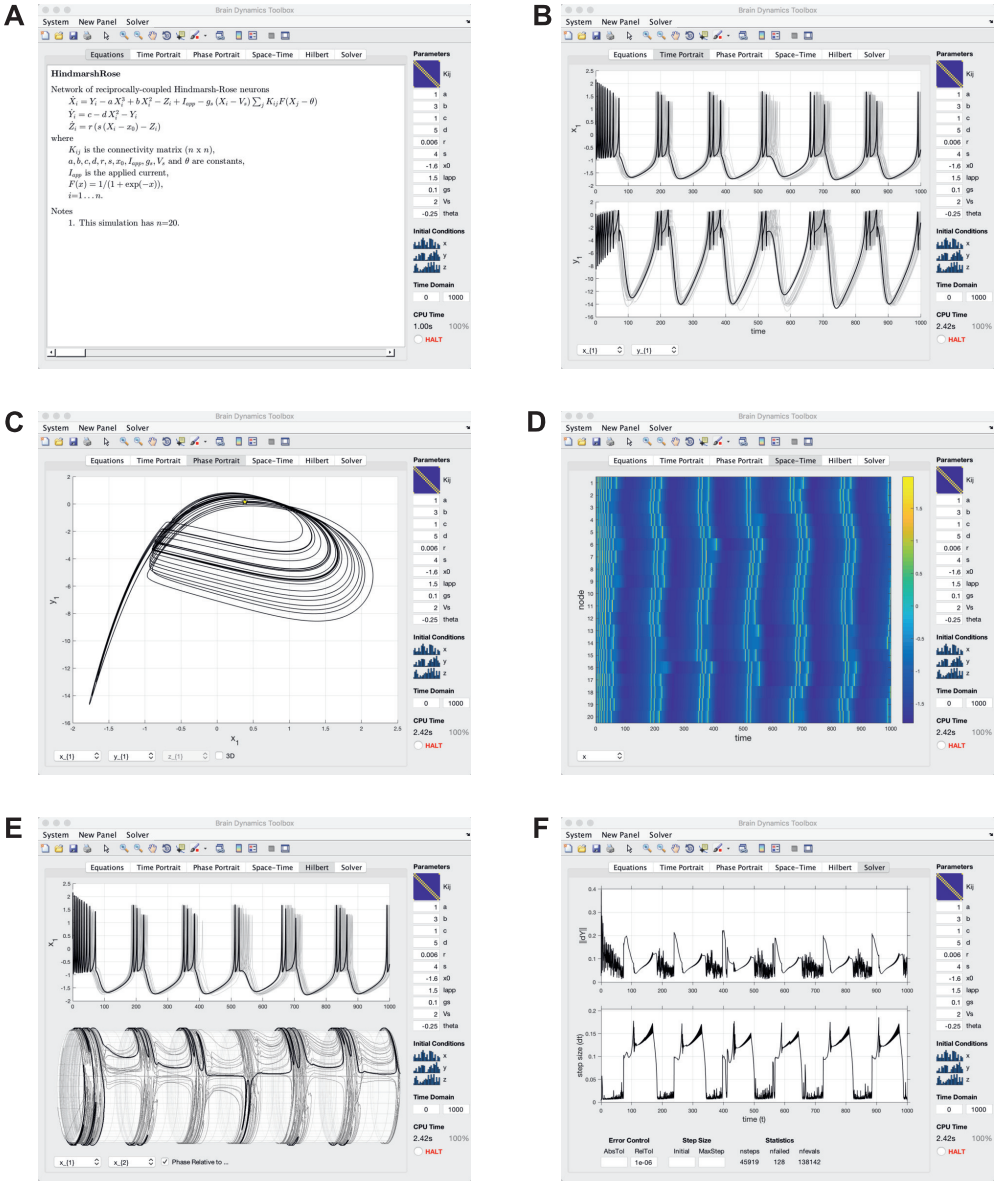

Figure 1: Screenshots of selected display panels in the graphical interface as it simulates a network of n=20 Hindmarsh-Rose [11] neurons. Display panels can be added or removed at run-time. The parameters of the model are manipulated with the control panel on the right-hand side of the application window. Individual controls can represent scalar, vector or matrix values. **A** The mathematical equations panel. **B** Time portraits. **C** Phase portrait. **D** Space-time portrait. **E** Hilbert transform. **F** Solver step sizes.

toolbox uses the same approach as the standard MATLAB solvers (e.g. `ode45`, `dde23`) except that it also manages the input parameters and plots the solver output. To do so, it requires the names and values of the system parameters and state variables. Those details (and more) are passed to the toolbox via a special data structure that we call a *system structure*. It encapsulates everything needed to simulate a user-defined model. Once a system structure has been constructed, it can be shared with other toolbox users.

3.1. Software Architecture and Functionality

The architecture is modular so that solver routines and visualization tools can be applied to any model in any combination (Figure 2). The combinatorial power of this approach brings great efficiency to the exploration of new models, as well as fostering the long-term evolution of the toolbox itself. The list of numerical solver routines and graphical panels that the toolbox supports continues to grow rapidly. As of this writing, it includes the standard ODE solvers (`ode45, ode23, ode113, ode15s, ode23s`) and DDE solver (`dde23`) that are shipped with MATLAB. As well as a fixed-step ODE solver (`odeEul`) and two SDE solvers (`sdeEM, sdeSH`) that are specific to the Brain Dynamics Toolbox. The two SDE solvers are specialized for stochastic equations that use Itô calculus and Stratonovich calculus respectively. Custom solvers can also be added provided that they adhere to toolbox conventions.

The display panels include time plots, phase portraits, space-time plots, linear correlations, Hilbert transforms, surrogate data transforms, solver statistics and mathematical equations rendered with LaTeX. New panels are being added on a regular basis and we encourage users to write custom panels for their own projects. Display panels not only visualize the solver output but can be used to apply transformations or custom metrics to it. The panel outputs themselves are accessible to the user as workspace variables where they can be readily saved for further analysis or publication.

4. Illustrative Example

We demonstrate the implementation of a network of recurrently-connected Hindmarsh-Rose [11] neurons,

$$\dot{X}_i = Y_i - a\,X_i^3 + b\,X_i^2 - Z_i + I_i - J_i, \tag{1}$$

$$\dot{Y}_i = c - d\,X_i^2 - Y_i, \tag{2}$$

$$\dot{Z}_i = r\,(s\,(X_i - x_0) - Z_i), \tag{3}$$

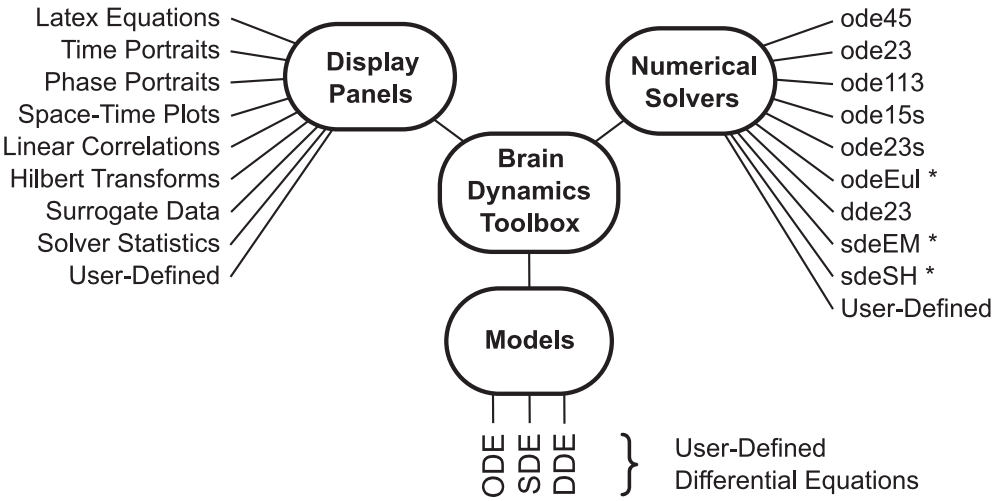

Figure 2: Software architecture of the Brain Dynamics Toolbox. Display panels, numerical solver routines and user-defined models are all implemented separately as inter-changeable modules. New models can thus be explored quickly with existing tools. Custom tools can also be defined for specialized investigations. Numerical solvers marked with an asterisk are unique to the toolbox.

where X_i is the membrane potential of the i^{th} neuron, Y_i is the conductance of that neuron's excitatory ion channels, and Z_i is the conductance of its inhibitory ion channels. Each neuron in the network is driven by a locally applied current I_i and a network current $J_i = g_s\,(X_i - V_s)\sum_j K_{ij} F(X_j - \theta)$ that represents the synaptic bombardment from other neurons. The sigmoidal function $F(x)=1/(1+\exp(-x))$ transforms that synaptic bombardment to an equivalent ionic current. The connectivity matrix K_{ij} defines the weightings of the synaptic connections between neurons. All other parameters in the model are scalar constants although their meaning is unimportant here. Suffice to say that this model represents a typical example of using coupled ODEs to model a neuronal network.

4.1. Defining the model

Listing 1 shows the code required to implement equations (1-3) with the toolbox. It consists of three functions: The main function, `HindmarshRose`, constructs the model's system structure (`sys`) for a given connectivity matrix, `Kij`. The `odefun` function defines the right-hand side of the differential equations (1–3). All but the first two input parameters of that function correspond to the ODE parameters (i.e. `Kij`, `a`, `b`, ...) as defined in the system structure (lines 8–20). The third function (lines 65–68) defines the sigmoid function used by this particularly model. It has no special significance to the toolbox.

Listing 1: Implementation of a network of Hindmarsh-Rose neurons (equations 1–3). The `HindmarshRose(Kij)` function takes a connectivity matrix `Kij` as input and returns a corresponding system structure for the model. The number of neurons (equations) in the model is determined from the size of `Kij` which is fixed for the life of the system structure.

```
1  function sys = HindmarshRose(Kij)
2      % determine the number of nodes from Kij
3      n = size(Kij,1);
4
5      % Handle to our ODE function
6      sys.odefun = @odefun;
7
8      % Our ODE parameters
9      sys.pardef = [ struct('name','Kij',    'value',Kij);
10                    struct('name','a',      'value',1);
11                    struct('name','b',      'value',3);
12                    struct('name','c',      'value',1);
13                    struct('name','d',      'value',5);
14                    struct('name','r',      'value',0.006);
```

```
                    struct('name','s',      'value',4);
                    struct('name','x0',     'value',-1.6);
                    struct('name','Iapp',   'value',1.5);
                    struct('name','gs',     'value',0.1);
                    struct('name','Vs',     'value',2);
                    struct('name','theta',  'value',-0.25) ];

    % Our ODE variables
    sys.vardef = [ struct('name','x', 'value',rand(n,1));
                   struct('name','y', 'value',rand(n,1));
                   struct('name','z', 'value',rand(n,1)) ];


    % Latex (Equations) panel
    sys.panels.bdLatexPanel.title = 'Equations';
    sys.panels.bdLatexPanel.latex = {
        '\textbf{HindmarshRose}';
        '';
        'Network of coupled Hindmarsh-Rose neurons';
        '\qquad $\dot X_i = Y_i - a\,X_i^3 + b\,X_i^2 - Z_i +
    I_{app} - g_s\,(X_i-V_s) \sum_j K_{ij} F(X_j-\theta)$';
        '\qquad $\dot Y_i = c - d\,X_i^2 - Y_i$';
        '\qquad $\dot Z_i = r\,(s\,(X_i-x_0) - Z_i)$';
        'where';
        '\qquad $K_{ij}$ is the connectivity matrix,';
        '\qquad $a, b, c, d, r, s, x_0, I_{app}, g_s, V_s$
   and $\theta$ are constants,';
        '\qquad $I_{app}$ is the applied current,';
        '\qquad $F(x) = 1/(1+\exp(-x))$,';
        '\qquad $i{=}1 \dots n$.' };
end

% The ODE function for the Hindmarsh Rose model.
function dY = odefun(t,Y,Kij,a,b,c,d,r,s,x0,I,gs,Vs,theta)
    % extract incoming variables from Y
    Y = reshape(Y,[],3);        % reshape Y to (nx3)
    x = Y(:,1);                 % x is (nx1) vector
    y = Y(:,2);                 % y is (nx1) vector
    z = Y(:,3);                 % z is (nx1) vector

    % The network coupling term
    Inet = gs*(x-Vs) .* (Kij*F(x-theta));

    % Hindmarsh-Rose equations
    dx = y - a*x.^3 + b*x.^2 - z + I - Inet;
```

```
    dy = c - d*x.^2 - y;
    dz = r*(s*(x-x0)-z);

    % return result (3n x 1)
    dY = [dx; dy; dz];
end

% Sigmoid function
function y=F(x)
    y = 1./(1+exp(-x));
end
```

The bulk of the code is dedicated to constructing the system structure (lines 1–43) which is described in detail in the *Handbook for the Brain Dynamics Toolbox* [12]. Among other things, the system structure requires a handle to the ODE function (line 6) as well as the names and values of the ODE parameters (lines 8–20) and the ODE variables (lines 22–25). It also defines the latex strings for rendering the mathematical equations in the display panel (lines 28–42). Those LaTeX strings are important for documenting the model but they play no part in the simulation itself. It is not unusual for LaTeX strings to be larger than the differential equations they describe.

4.2. Running the model.

The model is run by constructing an instance of its system structure and loading that into the toolbox graphical user interface, which is called `bdGUI`.

```
>> n = 20;                          % Define number of neurons.
>> Kij = circshift(eye(n),1) ...    % Define connection matrix,
       + circshift(eye(n),-1);      % as a chain in this case.
>> sys = HindmarshRose(Kij);        % Construct the sys struct.
>> bdGUI(sys);                      % Run the model in the GUI.
```

The graphical user interface (Figure 1) automatically recomputes the solution whenever any of the controls are adjusted. That includes the parameters of the model, the initial conditions of the state variables, the time domain of the simulation and the solver options. The computed solution can be visualized with any number of display panels, all of which are updated concurrently. The display panels in Figure 1 are from this very model.

4.3. Controlling the model

The `bdGUI` application returns a handle to itself which can be used to control the simulation from the MATLAB command window.

```
>> gui = bdGUI(sys)
gui =
  bdGUI with properties:
    version: '2017c'           % toolbox version string
        fig: [1x1 Figure]      % application figure handle
        par: [1x1 struct]      % system parameters (read-write)
       var0: [1x1 struct]      % initial conditions (read-write)
        var: [1x1 struct]      % solution variables (read-only)
          t: [1x9522 double]   % solution time points (read-only)
        lag: []                % DDE lag parameters (read-write)
        sys: [1x1 struct]      % system structure (read-only)
        sol: [1x1 struct]      % solver output (read-only)
        sox: []                % auxiliary variables (read-only)
     panels: [1x1 struct]      % display panel outputs (read-only)
```

The parameters of the model are accessible by name via the `gui.par` structure. Likewise, the computed solution variables are accessible by name via the `gui.var` structure. The output of the solver is also accessible in its native format via the `gui.sol` structure. The parameters and the initial conditions are read-write properties whereas the computed solutions are read-only. Any value written into the `gui` handle is immediately applied to the graphical user interface, and vice versa. Thus it is possible to use workspace commands to orchestrate parameter sweeps in the graphical user interface. For example,

```
>> for r=linspace(0.05,0.001,25); gui.par.r=r; end;
```

sweeps the r parameter (time constant of inhibition) from 0.05 to 0.001 in 25 increments. The computed solution is updated at each increment and a bursting phenomenon is observed for $r \lesssim 0.01$.

4.4. Scripting the model

The toolbox provides a small suite of command-line tools for running models without invoking the graphical interface. Of these, the most notable commands are `bdSolve(sys,tspan)` — which runs the solver on a given model for a given time span — and `bdEval(sol,t)` which interpolates the solution for a given set of time points.

```
>> t = 0:1000;
>> sol = bdSolve(sys,[t(1) t(end)]);
>> X = bdEval(sol,t);
>> plot(t,X);
```

The `bdEval` function is equivalent to the MATLAB `deval` function except that it also works for solution structures (`sol`) returned by third-party solvers. See the *Handbook for the Brain Dynamics Toolbox* [12] for a complete description of the command-line tools.

5. Conclusions

The Brain Dynamics Toolbox provides researchers with an interactive graphical tool for exploring user-defined dynamical systems without the burden of programming bespoke graphical applications. The graphical interface imposes no limit the size of the model nor the number of parameters involved. System parameters and variables can range in size from simple scalar values to large-scale vectors or matrices without loss of generality. The design also imposes no barrier to scripting large-scale simulations and parameter surveys. The toolbox is aimed at students, engineers and researchers in computational neuroscience but it can also be applied to general problems in dynamical systems. Once a model is implemented, it can be readily shared with other toolbox users. The toolbox thus serves as a hub for sharing models as much as it serves as a tool for simulating them. Indeed, we anticipate the number of available models and plotting tools to continue to grow as the user base expands.

Acknowledgements

MATLAB® is a registered trademark of The Mathworks, Inc., 3 Apple Hill Drive, Natick, MA 01760-2098 USA, 508-647-7000, Fax 508-647-7001, *info@mathworks.com, www.mathworks.com*

[1] J. M. Bower, D. Beeman, The book of Genesis: exploring realistic neural models with the General Neural Simulation System., Telos, Springer, New York, 1998.

[2] N. T. Carnevale, M. L. Hines, The NEURON book, Cambridge University Press, 2006.

[3] D. F. M. Goodman, R. Brette, BRIAN simulator, Scholarpedia 8 (1) (2013) 10883. doi:10.4249/scholarpedia.10883.

[4] V. Jirsa, O. Sporns, M. Breakspear, G. Deco, A. R. McIntosh, Towards the virtual brain: network modeling of the intact and the damaged brain, Archives Italiennes de Biologie 148 (3) (2010) 189–205.

[5] V. K. Jirsa, H. Haken, Field theory of electromagnetic brain activity, Phys. Rev. Lett. 77 (5) (1996) 960.

[6] E. J. Doedel, A. R. Champneys, T. F. Fairgrieve, Y. A. Kuznetsov, B. Sandstede, X. Wang, AUTO 97: Continuation and bifurcation software for ordinary differential equations (with HomCont) (1998).

[7] B. Ermentrout, Simulating, analyzing, and animating dynamical systems: a guide to XPPAUT for researchers and students, SIAM, 2002.

[8] A. Dhooge, W. Govaerts, Y. A. Kuznetsov, MATCONT: a MATLAB package for numerical bifurcation analysis of ODEs, ACM Trans Math Softw 29 (2) (2003) 141–164.

[9] R. Clewley, Hybrid Models and Biological Model Reduction with PyDSTool, PLOS Computational Biology 8 (8) (2012) e1002628. doi:10.1371/journal.pcbi.1002628.

[10] H. Dankowicz, F. Schilder, Recipes for Continuation, SIAM, 2013.

[11] J. L. Hindmarsh, R. M. Rose, A model of neuronal bursting using three coupled first order differential equations, Proceedings of the Royal Society of London B: Biological Sciences 221 (1222) (1984) 87–102.

[12] S. Heitmann, M. Breakspear, Handbook for the Brain Dynamics Toolbox: Version 2017c, QIMR Berghofer Medical Research Institute, 2017.

Required Metadata

Current executable software version

Nr.	(executable) Software metadata description	Please fill in this column
S1	Current software version	2017c
S2	Permanent link to executables of this version	https://github.com/breakspear/bdtoolkit/releases/tag/bdtoolkit-2017c
S3	Legal Software License	BSD 2-clause
S4	Computing platform/Operating System	Matlab 2014b or newer
S5	Installation requirements & dependencies	Signal Processing Toolbox (optional). Statistics and Machine Learning Toolbox (optional).
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	https://bdtoolkit.blogspot.com
S7	Support email for questions	stewart.heitmann@gmail.com

Table 1: Software metadata (optional)

Current code version

Nr.	**Code metadata description**	**Please fill in this column**
C1	Current code version	2017c
C2	Permanent link to code/repository used of this code version	https://github.com/breakspear/bdtoolkit/releases/tag/bdtoolkit-2017c
C3	Legal Code License	BSD 2-clause
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Matlab 2014b or newer
C6	Compilation requirements, operating environments & dependencies	Signal Processing Toolbox (optional). Statistics and Machine Learning Toolbox (optional).
C7	If available Link to developer documentation/manual	https://bdtoolkit.blogspot.com
C8	Support email for questions	stewart.heitmann@gmail.com

Table 2: Code metadata (mandatory)