Brain Dynamics Toolbox

Version 2017b

The Brain Dynamics Toolbox provides a convenient graphical user interface for exploring dynamical systems in MATLAB. Users implement their own dynamical equations (as matlab scripts) and use the toolbox graphical interface to view phase portraits and other plots in real-time. The same models can also be run as MATLAB scripts without the graphics interface. The toolbox includes solvers for Ordinary Differential Equations (ODE), Delay Differential Equations (DDE) and Stochastic Differential Equations (SDE). The plotting tools are modular so that users can create custom plots according to their needs. Custom solver routines can also be used. The user interface is designed for dynamical systems with large numbers of variables and parameters, as is often the case in dynamical models of the brain. Hence the name, *Brain Dynamics Toolbox*.

Download

Dowload the latest release from the bdtoolkit repository on GitHub

Getting Started

The toolbox requires MATLAB 2014b or newer. Unzip the toolbox files into a directory of your choosing. The main toolbox scripts are located in the *bdtoolkit* directory which must be in your matlab PATH variable. The *bdtoolkit/models* directory contains example scripts that are also advisable to have in your PATH.

```
$ unzip bdtoolkit-2017b.zip
$ matlab
>> addpath bdtoolkit
>> addpath bdtoolkit/models
```

Each dynamical system (model) is defined by a specially formatted data structure that we call a *sys* struct. An existing *sys* struct can be loaded from a *mat* file or a new one can be constructed from a model-specific script. The *models* directory contains both scripts and *mat* files for numerous dynamical systems. The following example shows how to load the pre-defined *sys* struct for the Hindmarsh-Rose model from a *mat* file into the graphical toolbox (bdGUI).

```
>> load HindmarshRose.mat sys      % load the sys struct from file
>> bdGUI(sys);                     % run the graphic user interface
```

The toolbox allows the user to vary the model's parameters and solve the dynamical equations by forwardintegration. However some aspects of any model are fixed at construction, such as the number of neurons in the Hindmarsh-Rose model. The next example shows how to use the HindmarshRose.m script to construct a *sys* struct for a network of 242 neurons using a connectivity matrix (MacCrtx) from the CoCoMac connectome database (cocomac242.mat).

```
>> load cocomac242.mat MacCrtx    % load the connectivity matrix from file
>> sys = HindmarshRose(MacCrtx);  % construct a sys struct for the model
>> bdGUI(sys);                    % run the graphic user interface
```

This final example shows how to construct a system of n=21 randomly connected Hindmarsh-Rose neurons with a user-defined connectivity matrix (Kij).

```
>> help HindmarshRose             % model-specific help
>> n = 21;                        % number of neurons
>> Kij = rand(n);                 % random connectivity matrix (nxn)
>> sys = HindmarshRose(Kij);      % construct the sys struct
>> bdGUI(sys);                    % run the graphic user interface
```

Many of the model scripts shipped with the toolkit follow this same basic approach: The script determines the size of the model (number of dynamical equations) from some input parameter (such as a connectivity matrix) and then returns a suitable *sys* struct for use with the graphic toolbox. However each script is unique. Use the *help* function for the details of each model.

How it works

The toolkit uses standard ODE, DDE and (new) SDE solvers to integrate (solve) a set of dynamical equations provided by the user. The user supplies the right hand side of their dynamical equation as a Matlab function, in the same way as is done for the Matlab *ode45* solver. That user-supplied function and additional information about the dynamical system (parameter names, variable names, solver options, plotting options) are encapsulated within a special structure known as the *sys struct.* It contains everything that the toolbox needs to know about solving and plotting the dynamical system.

A typical sys struct has the following structure:

```
% Handle to a user-defined ODE function
sys.odefun = @odefun;

% ODE parameter definitions
sys.pardef = [ struct('name','a', 'value',-1);          % a = -1
               struct('name','b', 'value',0.01) ];      % b = 0.01

% ODE state variable definitions
sys.vardef = struct('name','y', 'value',0.9);           % y = 0.9

% Time span of the solution
sys.tspan = [0 5];

% ODE solver options
sys.odeoption.AbsTol = 1e-3;
sys.odeoption.RelTol = 1e-6;

% Time Portrait options
sys.panels.bdTimePortrait.title = 'Time Portrait';
sys.panels.bdTimePortrait.grid = True;

% Phase Portrait options
sys.panels.bdPhasePortrait.title = 'Phase Portrait';
sys.panels.bdPhasePortrait.vecfield = True;
```

The *sys.odefun* field is a function handle to a user-defined function of the form:

```
function dYdt = odefun(t,Y,a,b)
   dYdt = a*Y + b*t;
end
```

Display Panels

The plotting tools (display panels) are modular by design so that custom panels may be written by the user. The standard panels are located in the *bdtoolkit/panels* directory. The list of standard panels continues to grow. It currently includes time portraits, phase portraits, space-time plots, linear correlation, Hilbert transforms, latex equations and solver statistics. The user may load any of these panels into the toolbox at run-time. The toolbox also pre-loads those panels that are listed in the model's *sys.panels* options. A typical example is the *bdLatexPanel* which displays mathematical equations using latex. The following example is from *LinearODE.m* which implements a coupled pair of linear Ordinary Differential Equations.

```
    sys.panels.bdLatexPanel.title = 'Equations';
    sys.panels.bdLatexPanel.latex = {
        '\textbf{LinearODE}';
        '';
        'System of linear ordinary differential equations';
        '\qquad $\dot x(t) = a\,x(t) + b\,y(t)$';
        '\qquad $\dot y(t) = c\,x(t) + d\,y(t)$';
        'where $a,b,c,d$ are scalar constants.';
        };
```

Useful utilities

The `bdSysCheck` function is a helpful tool for validating the system structure of a new model. It checks that the various fields of the sys struct are properly defined. It also tests the user-defined function handle(s) to verify that they return data in the proper format. It is recommended that custom models be routinely checked with *bdSysCheck* during development.

The `bdSolve` function solves a user-supplied model without invoking the graphic user interface. It is useful for batch processing. The `bdSetValue` and `bdGetValue` functions are also useful for setting and getting the values of *sys.pardef* and *sys.vardef* data structures in batch scripts.

The `bdLoadMatrix` function is useful for loading matrix data from a file.

The `bdEditScalars`, `bdEditVector` and `bdEditMatrix` functions are useful for interactively editing scalars, vectors and matrices, respectively.

The `Trap Panel` is useful for detecting erroneous drawing commands by user-defined GUI panels.

Going Further

A user manual is currently being written. Until that is published, the best way to proceed is to browse the example code in the *models* directory.

Ordinary Differential Equations:

- *BTF2003ODE* - Breakspear, Terry & Friston (2003) Neural mass model.
- *FitzhughNagumo* - Generalized FitzHugh-Nagumo neural oscillator.
- *HindmarshRose* - Network of Hindmarsh-Rose neurons.
- *HopfieldNet* - Hopfield Network.
- *KuramotoNet* - Network of Kuramoto phase oscillators.
- *LinearODE* - A simple Ordinary Differential Equation.
- *SwiftHohenberg1D* - Discretised Swift-Hohenberg PDE in one spatial dimension.
- *VanDerPolOscillators* - Network of Van Der Pol oscillators.

- *WaveEquation1D* - Discretized wave equation (PDE) in one spatial dimension.

Delay Differential Equations:

- *BTF2003DDE* - Breakspear, Terry & Friston (2003) Neural mass model with time delays.
- *WilleBaker* - A simple Delay Differential Equation.

Stochastic Differential Equations:

- *BrownianMotion* - Geometric Brownian motion.
- *BTF2003SDE* - Breakspear, Terry & Friston (2003) Neural mass model with multiplicative noise.
- *FRRB2012* - Freyer, Roberts, Ritter & Breakspear (2012) Canoncial subcritical Hopf model with multiplicative noise.
- *FRRB2012b* - Freyer, Roberts, Ritter & Breakspear (2012) Network variant of FRRB2012.
- *MultiplicativeNoise* - A simple example of multiplicative noise.
- *Ornstein Uhlenbeck* - Multiple independent Ornstein-Uhlenbeck processes.
- *RFB2017* - Roberts, Friston & Breakspear (2017) Isolated neural mass with multiplicative noise.

BSD License

Contributors

- Michael Breakspear, Project Leader
- Stewart Heitmann, Project Leader, Lead Developer
- Matthew Aburn