# Brain Dynamics Toolbox

*Version 2017a*

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 top level of the *bdtoolkit* directory. The solver routines (*solvers*) and plotting tools (*panels*) are located in their own subdirectories. The *models* subdirectory contains example dynamical systems. All of these directories should be in your matlab PATH variable. You may then run the *bdGUI* application and load one of the pre-defined models (eg HindmarshRose.mat) using the *System-Load* menu.

```
>> addpath bdtoolkit
>> addpath bdtoolkit/solvers
>> addpath bdtoolkit/panels
>> addpath bdtoolkit/models
>> bdGUI
Open bdtoolkit/models/HindmarshRose.mat
```

## How it works

The brain dynamics toolkit uses the Matlab ODE and DDE solvers to integrate (solve) a set of dynamical equations provided by the user. The user supplies the right hand side of the dynamical equation as a function in the same way they do for *ode45*. The difference is that the user-supplied function, as well as

additional information about the dynamical system (parameter names, variable names, etc), are encapsulated within a special structure known as the *system struct*. It contains everything that the toolbox needs to know about solving and plotting the dynamical system.

A typical system struct has the following fields:

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

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

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

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

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

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
```

# Models

The toolkit ships with a collection of pre-defined models in the *bdtoolkit/models* directory. Each model is a matlab script that returns a *sys* struct which the user then loads into the graphical user interface (bdGUI). Use the matlab HELP function for the syntax of each script.

```
>> help LinearODE      % get help on the LinearODE model.
>> sys = LinearODE();  % construct the system structure.
>> gui = bdGUI(sys);   % load it into the toolkit GUI.
```

# Panels

The default plotting tools (panels) are loaded by the GUI in accordance with the model's `sys.panels` options. Those options corespond to the names of the panel classes which are located in the *bdtoolkit/panels* directory. The code snippet below gives an example.

```
    % Options for the Time Portrait panel
    sys.panels.bdTimePortrait.title = 'Time Portrait';
    sys.panels.bdTimePortrait.grid = True;

    % Options for the Phase Potrait panel
    sys.panels.bdPhasePortrait.title = 'Phase Portrait';
    sys.panels.bdPhasePortrait.vecfield = True;
```

The user can always load new panels at run-time so not all panels need to be predefined in the *sys* structure. Nontheless doing so is beneficial when model-specific options are involved. A common example is the *bdLatexPanel* which is used to display the relevant mathematical equations using latex. The following example is from *LinearODE.m*.

```
    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. Any new model should be tested with *bdSysCheck* as standard practice.

The `bdSolve` function solves a user-supplied model without invoking the graphic user interface. It is useful for batch processing. Likewise, the `bdSetValue` and `bdGetValue` functions provide convenient methods to set and get the values of *sys.pardef* and *sys.vardef* data structures from within 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.

# Going Further

The best way to proceed is to inspect the example code in the *models* directory. Several introductory

models are provided: (i) *LinearODE* illustrates a simple Ordinary Differential Equation; (ii) *DDEdemo1* illustrates a simple Delay Differential Equation; (iii) *MultiplicativeNoise* illustrates a simple Stochastic Differential Equation.

# BSD License



# Contributors

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