Brain Dynamics Toolbox

Version 2016a

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*.

Getting Started

The toolbox requires MATLAB 2014b or better. 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
```

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 graphic user interface needs to know about the dynamical system.

A typical system struct has the following fields:

```
sys.odefun = @odefun;         % Handle to our ODE function
sys.pardef = {'a',-1;         % ODE parameters {'name', value}
              'b',0.01};
sys.vardef = {'y',rand};      % ODE variables {'name',value}
sys.tspan = [0 5];            % solution time span
sys.odesolver = {@ode45,      % matlab ODE solvers
                 @ode23}
sys.odeoption.RelTol = 1e-6;  % solver options
sys.odeoption.AbsTol = 1e-6;  % same as odeset
```

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

```
% A simple ODE function
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 constructed using a matlab script that returns a *sys* struct. That *sys* struct is then passed to the graphical user interface (bdGUI). Different models have different script parameters, so use the matlab HELP function to see how each script should be used.

```
>> help ODEdemo1             % get help on using the model
>> sys = ODEdemo1();         % construct the model as a sys struct
>> gui = bdGUI(sys);         % pass the model to the toolkit GUI application
```

Panels

The plotting tools (panels) are loaded by the GUI in accordance with the contents of `sys.gui` in the model's system structure. The top-level field names correspond to the classes defined in the *bdtoolkit/panels* directory. The following snippet tells the bdGUI application to load the *bdTimePortrait*, *bdPhasePortrait* and *bdSolver* panels.

```
sys.gui.bdTimePortrait.title = 'Time Portrait';
sys.gui.bdPhasePortrait.title = 'Phase Portrait';
sys.gui.bdSolverPanel.title = 'Solver';
```

The *bdLatexPanel* is typically the first panel loaded by any model. It uses *latex* to display the relevant mathematical equations. The user defines the relevant latex code in *sys.gui.bdLatexPanel.latex* as a cell array of strings. Each cell corresponds to one line of latex output. The following example is from *ODEdemo1.m*.

```
    sys.gui.bdLatexPanel.title = 'Equations';
    sys.gui.bdLatexPanel.latex = {'\textbf{ODEdemo1}';
        '';
        'An Ordinary Differential Equation (ODE)';
        '\qquad $\dot Y(t) = a\,Y(t) + b\,t$';
        'where $a$ and $b$ are scalar constants.'};
```

Useful utilities

The `bdVerify(sys)` 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 *bdVerify* as standard practice.

The `bdSolve(sys)` function solves a user-supplied model without invoking the graphic user interface. It is useful for batch processing.

The `bdSetValue(xxxdef,name,val)` function provides a convenient method to set the value of parameter in any of the *pardef*, *vardef* or *lagdef* cell arrays in a *sys* struct.

Likewise, `bdGetValue(xxxdef,name)` provides a convenient method to read a value from a *pardef*, *vardef* or *lagdef* cell array.

The `bdLoadMatrix(name,msg)` function is useful for loading matrix data from a matlab file or importing it from a data file of another format.

The `bdEditScalars(pardef,name,descr)`, `bdEditVector(indata,name,columnName)` and `bdEditMatrix(inata,name)` functions are useful for interactively editing scalars, vectors and matrix data respectively.

Going Further

The best way to proceed is to inspect the example code in the *models* directory. In particular, *ODEdemo1* demonstrates an introductory Ordinary Differential Equation. Likewise, *DDEdemo1* demonstrates an introductory Delay Differential Equation and *SDEdemo1* deomstrates an introductory Stochastic Differential Equation

BSD License

This software is freely available under the 2-clause BSD license.

> Redistribution and use in source and binary forms, with or without modification, are permitted provided

Contributors

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