SCI131 Code Guide

Computing Basin Boundaries in Julia

Mahnoor Qadri
Supervised by Claire Postlethwaite and Matthew Egbert

Contents

1 Getting Started

1.1 Downloading Julia

Download the relevant repository from GitHub: `https://github.com/s-mahnoor-q/viabilitysystem`
Download Julia v1.10.0: `https://julialang.org/downloads/#current_stable_release`

Originally I was using VSCode, but it has issues with the readline() function, which makes input frustrating, so we will be working with the command terminal instead.

> **Note: This code has been developed using Windows. The Julia code should work just fine, but code handling input/output may need to be adjusted as it has not been tested on other OS.**

1.2 Setup

Open the clone of the github repository in your file explorer, and open the command terminal. Using the directory of your local github project folder, `julia --project= /viabilitysystem` to open Julia and load all packages. In order to check if the environment is loaded and the version of the packages, enter the command `] status`. At the time of writing, this is the following set of packages used in the environment [1][2][3][4][5]:

```
(viabilitysystem) pkg> status
Status `C:\Users\smahn\OneDrive\Documents\Github Repos\viabilitysystem\
    Project.toml`
  [f3fd9213] Attractors v1.13.6
  [0f109fa4] BifurcationKit v0.3.3
  [0c46a032] DifferentialEquations v7.12.0
  [61744808] DynamicalSystems v3.2.4
  [e9467ef8] GLMakie v0.9.5
  [1ecd5474] GraphMakie v0.5.9
  [86223c79] Graphs v1.9.0
  [033835bb] JLD2 v0.4.45
  [ee78f7c6] Makie v0.20.4
```

1.3 Precompilation

In order to have faster execution time, it is recommended to run `] precompile`. The first execution of scripts may be slow as well, due to more precompilation of a lot of packages, but after that, execution speed of generating networks and loading the applet takes at *most* 2 seconds.

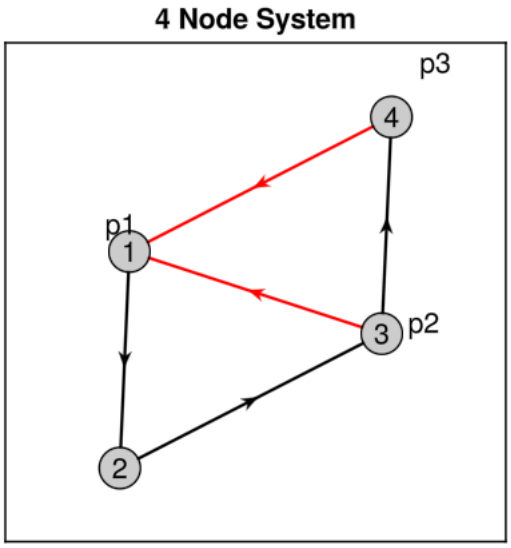
2 Using the program

2.1 Params vs Input Function?

There are two separate versions of the applet and network generating file. This is because currently, you can either create a system which has constant input parameters, or one parameter which is a sine wave.

2.2 `generate_network_v3_params.jl`

Open the Julia v1.10.0. To execute any script in the same folder as `pwd()`, type `include("filename.jld2")`. If the script is in a different folder, you will need to adjust directories. The commands required to create the following network are outlined below:




```
include("generate_network_v3_param.jl")
Welcome to the network generator!
--------------------------------
Note that entries with 1 will be AND together, and 2 will be OR together
Please select the number of nodes in your network:
4
Please enter the adjacency row of node 1, separated by spaces:
0 1 0 0
Please enter the adjacency row of node 2, separated by spaces:
0 0 1 0
Please enter the adjacency row of node 3, separated by spaces:
2 0 0 1
Please enter the adjacency row of node 4, separated by spaces:
2 0 0 0
Adjacency matrix:
[0 1 0 0; 0 0 1 0; 2 0 0 1; 2 0 0 0]
Please enter the parameters of your network, separated by spaces:
0.7 0.8 0.5
Entering parameter adjacency matrix: e.g. parameter 1 feeds into node 1 and
    2, so has adjacency row [1 1 0 ...]
Please enter the adjacency row of parameter 1 = 0.7, separated by spaces:
```

```
1 0 0 0
Please enter the adjacency row of parameter 2 = 0.8, separated by spaces:
0 0 1 0
Please enter the adjacency row of parameter 3 = 0.5, separated by spaces:
0 0 0 1
Parameters: [0.7, 0.8, 0.5]
Please enter the index of the parameter to bifurcate:
1
Index: 1
Please enter the range to bifurcate from and to separated by a space:
-0.5 1.5
*Please enter the initial conditions of your network, separated by spaces:
0.4 0.5 0.9 0.4
Would you like to save this network? (y/n)
y
Please enter the name of the file you would like to save to (e.g. net.jld2):
fournode_test.jld2
File saved successfully!
ODENetworkModel(4, [0 1 0 0; 0 0 1 0; 2 0 0 1; 2 0 0 0], [1 0 0 0; 0 0 1 0; 0
     0 0 1], [0.7, 0.8, 0.5], 1, -0.5, 1.5, [0.4, 0.5, 0.9, 0.4])
```

2.3 appv7_params.jl

This script contains code which generates the UI of the applet, and generates relevant trajectories. Outlines of some key functions are given below.

`create_viability_model(odenetworkmodel::ODENetworkModel, u0 = odenetworkmodel.u0; return_eqs = false)`
This function generates a set of equations corresponding to the adjacency matrix provided. First, any nodes/variables which are connected by OR relations are added to a sum counter, then any nodes/variables which are connected by AND relations are multiplied to the sum, and finally any parameters are multiplied to the result. For some arbitrary node T_i with the following set of OR edges that connected to the nodes $\{N_1, N_2, ..., N_k\}$, a set of AND edges which connect to nodes $\{M_1, M_2, ..., M_p\}$, and a set of parameters which affect T_i, $\{p_1, ..., p_d\}$, we get the following differential equation for that variable:

$\dot{T_i} = -T_i + \Phi(p_1...p_d(M_1M_2...M_p)(N_1 + N_2 + ... + N_k))$,
where $\Phi(x) = [1 + exp(-\frac{(x-\theta)}{\epsilon})]^{-1}$.

This means that it is currently *not* possible to have an implementation of A dependent on (B AND (C OR D)) OR E AND F:
$\dot{A} = -A + \Phi(B(C + D) + EF)$
This may be added in later implementations, and will likely need more complicated adjacency matrices.

`create_UI(ODENetworkModel)`
This function creates the UI for the ODENetwork model. The layout is broken into several GridLayout containers:

- `network` - holds a directed graph representation of the network

- `textboxes_frame` - holds all the textboxes where you can enter initial conditions, the rerun button and randomise initial conditions button
- `timeseries_box` - holds the Axis which the timeseries is plotted onto
- `phase_space_box` - holds an Axis or Axis3 object (depending on whether the system has 2 nodes or mode), which is used to plot the trajectory onto
- `axis_opts` - for 3 dimensions, holds dropdown menus which change which variable is plotted along which axis
- `slider_box` - holds the sliders which change the parameters

Updating the variables internally whenever any of the UI is changed is done via `@lift begin ... end` or `on(x) do x ... end` blocks, which are called whenever the value stored in an Observable are changed. Commonly repeated short tasks as such are stored in function definitions inside `create_UI()`. For more information, see the page Observables and Interaction.

`create_attractors(ODENetworkModel)`

This function uses functions from `Attractors.jl` to find the attractors of a system of 3 or more nodes, using an ODENetworkModel input. This function is not included in the main GUI, and can be called from main() if needed.

`create_basins_2d(ODENetworkModel, def = 0.1, param = 0.5)`

This function is calculates and creates a figure of a two node system's basin diagram in a separate window. This is leftover from some simple visualisations that were created for a presentation.

`create_branch(model!, ax, u0, par, opts_br, p_index)`

This function is called in `create_bifurcation_diagram()`, and uses `BifurcationKit.continuation()` to create a branch and add it to the Axis.

`create_bifurcation_diagram(ODENetworkModel)`

Creates a bifurcation diagram for a given ODENetworkModel, but assumes that the behaviour is qualitatively the same as the two dimensional model elaborated on in Section 3.1 of the Summer Research Report. This is why continuation is implemented from an initial condition close to [0.0, ..., 0.0] and another close to [1.0, ..., 1.0]. Results are displayed in a separate window. In order to fully find all branches (currently not deemed necessary because the behaviour of connected networks is similar), this may be updated to use Deflated Continuation.

`main()`

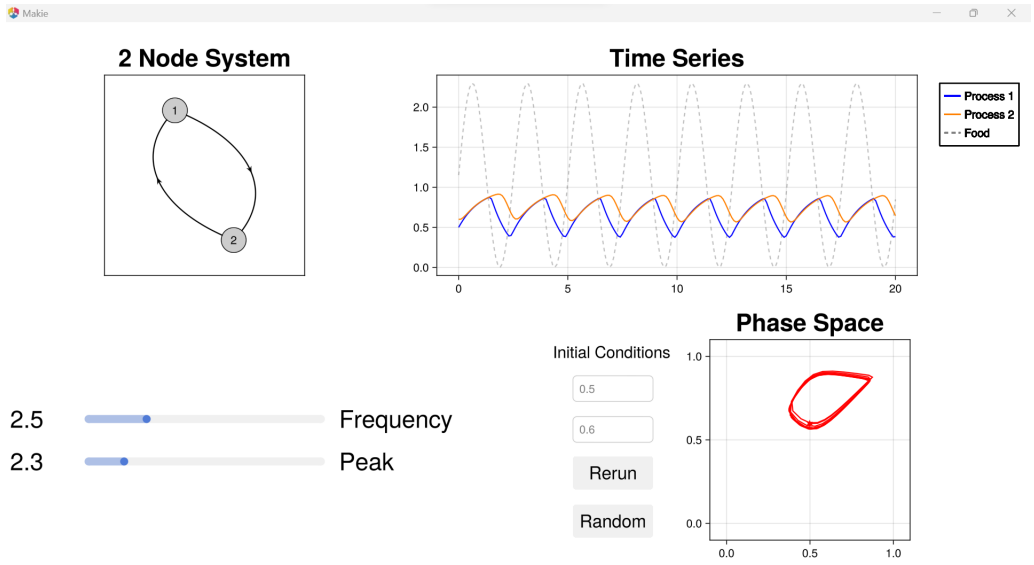
The main function uses command line output and input to determine which network should be loaded, and includes error detection. To exit the loop, use `Ctrl + C`.

2.4 `generate_network_v3_inputfunc.jl`

When run in the command terminal, this file will ask for the adjacency matrix of a network, but the main difference is that in the parameter section, it will ask for exactly two parameters. These are the initial (frequency, peak) of the sine wave, and you can choose which nodes this input function feeds into in the next prompt.

2.5 `appv7_inputfunc.jl`

This script is very similar to `appv7_params.jl`, except where there were previously sliders which changed the parameters, now there are only two sliders, which control the frequency and peak of the input sine wave. Additionally, the input function is plotted in a dotted grey line on the time series graph.



3 Limitations and Future Work

Currently the topologies that can be represented with the applet are limited, and through analysis which is further detailed in the summer research report, are similar in behaviour.

Future work may include the ability to edit the topology in UI by having a click-to-add feature for nodes or edges, and having a better representation of the parameter/food sources.

As well as this, the app could be optimised by looking into the best differential equation solver so that trajectories can be interactively calculated with minimised lag, while still having reasonable error tolerances.

References

[1] R. Veltz, "BifurcationKit.jl," Jul. 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02902346

[2] C. Rackauckas and Q. Nie, "DifferentialEquations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia," *Journal of Open Research Software*, vol. 5, no. 1, 2017.

[3] G. Datseris, "Dynamicalsystems.jl: A julia software library for chaos and nonlinear dynamics," *Journal of Open Source Software*, vol. 3, no. 23, p. 598, mar 2018. [Online]. Available: https://doi.org/10.21105/joss.00598

[4] S. Danisch and J. Krumbiegel, "Makie.jl: Flexible high-performance data visualization for Julia," *Journal of Open Source Software*, vol. 6, no. 65, p. 3349, 2021. [Online]. Available: https://doi.org/10.21105/joss.03349

[5] J. Fairbanks, M. Besançon, S. Simon, J. Hoffiman, N. Eubank, and S. Karpinski, "Juliagraphs/graphs.jl: an optimized graphs package for the julia programming language," 2021. [Online]. Available: https://github.com/JuliaGraphs/Graphs.jl/