Contents

1 Questions for Krasy

- How do you produce bifurcation diagrams on the surface of a sphere, like those in your paper?
- How do you find the slow subsystem paths? Is it just looking for the right order of bifurcations and testing them out through trial and error?
- Why are we looking at bifurcations on the surface of the sphere? Is it just as a convenient dimensionality reduction? Is there any a priori reason why this surface should have contained the burster path, or was that just a lucky bit of guesswork?
- How did you come up with the parameter values for the bifurcation regions?
  - Golubitsky uses b=3, mu2 = 1/3 for the cubic model; is there any a priori reason to choose this, or is it trial and error?
  - The paper fixes b=0.75. I loosely understand that as being a way of reducing the dimension of the parameter space, while still being close enough to the singularity to contain the interesting additional bifurcations it creates, but is there any particular intuition about why this is a useful parameter value? Why close to the origin? Why not really close to the origin? The paper claims that for small b, any fixed neighbourhood (here the unit ball) around the origin of (mu1,mu2,v) space contains all the additional bifurcations gained from the doubly-degenerate BT point. Why is this the case? How and why was b=0.75 chosen? And the unit ball?
- How biologically relevant is this model, being a phenomenological one? Will it be able to tell me much about what a living neuron is doing? How much information can we get using it to describe a living cell, if we don't consider the different ionic currents?
  - Is there any nice methods for relating the parameters and parameter values back to what's going on in the live cell? Any real-world interpretation of them? In the cubic Lienard model, mu1 is like a slow subsystem variable (z, in the Hindmarsh-Rose model), and v is the HR b-like bursting pattern variable, but even these variables are something of a phenomenological abstraction. Does this mean the model is only a useful test bed for in-silico CBC, or would it also be applicable to controller design?

2 Continuation software

2.1 Notes

2.1.1 ODE examples

1. XPP
2. COCO
3. MATCONT
4. PyDSTool

2.1.2 Pure AUTO capabilities

1. Algebraics
   - Compute sol'n families for algebraic eq's of form $f(u, p) = 0,\ f(\cdot, \cdot) \in \mathbb{R}^n$
   - Find branch points, and continue them in two or three parameters
   - Find Hopf points, continue them in two parameters, detect criticality, find zero-Hopf, BT, Bautins
   - Find folds, continue in 2 parameters, find cusps, zero-Hopfs, BTs
   - Find branch points, folds, period doubling, Neimark-Sackers, continue these in 2 or 3 params and switch branches at branch points and PD bifs for map fixed points
   - Find extrema of ojective functions along solution families; continue extrema in more params
2. Flows Consider an ODE of form $u'(t) = f\big(u(t), p\big),\ f(\cdot, \cdot),\ u(\cdot) \in \mathbb{R}^n$. AUTO can...
   - Compute stable / unstable periodic sol'n families, and their Floquet multipliers
   - Find folds, branch points, period doublings, Neimark-Sackers, along PO families; branch switching at PO and PD bifs
   - Continue folds, PD bifs, NS bifs in two parameters, and detect 1:{1,2,3,4} resonances
   - Continuation of fixed-period orbits for sufficiently large periods
   - Follow curves of homoclinic orbits, detect and continue codim-2 bifs using HomCont
   - Find extrema of integral objective functions along a periodic solution family; continue extrema in more parameters
   - Compute sol'n curves on the unit interval, subject to nonlinear BCs and integral conditions; discretisation uses an adaptive-mesh orthogonal collocation
   - Determine fold, branch points along sol'n families to the above BVP
3. PDEs Also some stuff for reaction-diffusion equations.

2.1.3 Things to put in the paper

Table of comparison:

- Bifurcations it can do, curves it can continue, and the types of system they can use
- When they fail, crash, etc.
- Numerical methods they have available
- How much do the parameters need manually fiddling?
- Do we need to code or not?

When writing, aim it at a biology audience. Continuation is a sequence of problems - start off at equilibria, then move to tracking codim2 bifurcations, increase the dimension etc. Make this nice and clear: explain why we're starting off finding any sorts of bifurcations we can, then continuing those to find others. Aim it at someone that doesn't understand continuation (assume they know what bifurcations are, but not continuation methods for finding them). A brief section on the maths (eg. why we need to continue from a steady state, and how continuation works) would probably be useful.

2.1.4 Investigating the HR model

1. Simplifying assumptions
    - b is a parameter influencing the bursting and spiking behaviour (frequency of spiking, ability or inability to burst)
    - We want to find the start/stop bifurcations when in a spiking regime, so we fix I=2 to force the neuron to spike
    - Freeze the fast subsystem (so, ignore the slow subsystem)
    - We therefore have two bifurcation parameters - slow subsystem state z, and bursting-spiking parameter b
2. Investigation strategy
    - Simulate the neuron for a few different b,z, to see what happens
    - It spikes
    - If the neuron can spike there must be a limit cycle; if there's a planar limit cycle, there must be an equilibrium within it
    - We're interested in when this limit cycle appears or disappears; let's start by investigating how its central equilibrium bifurcates

    (a) Equilibrium bifurcation (1) Find the equilibrium
    - Simulate the system to get a (x,y) phase portrait, for arbitrary initial conditions, params
        - Wikipedia says b=3 is a sensible value, so let's use that to start with
        - The simulations seem to show I=2 as being a nice (but arbitrarily chosen!) value, so let's use that too
        - (Emphasise that these were chosen just by playing around with simulations)

- This shows a stable limit cycle
- Choose some point within the limit cycle and integrate backwards
- This allows us to find the (unstable!) equilibrium in the middle of the limit cycle
  - For I=2, b=3, other params at wikipedia default, this gives an equilibrium at x,y=1,-4

(2) Do a bifurcation analysis in Z of this equilibrium

- We choose to bifurcate in Z since this is the forcing term applied by the slow subsystem that causes bursting
- Since we have a 1d slow subsystem, we must have a hysteresis-loop burster; hyseteresis-loops typically have a Z-shaped nullcline, so let's guess that's going to be the case and plot a bifurcation diagram in (z,x) space
- We get two LPs and two Hopf's; the first of these Hopfs occurs at z<-10; this is outside the expected range of z for a typical HR firing, so we'll ignore this one and focus on the other three bifs

(3) Continue the bifurcations in (z,b) space

- Get confused and give up?

2.1.5 Refs

[1] `http://www.math.pitt.edu/~bard/xpp/whatis.html`

[2] K. Engelborghs, T.Luzyanina, G. Samaey, DDE-BIFTOOL v. 2.00: a Matlab package for bifurcation analysis of delay differential equations, Technical Report TW-330, Department of Computer Science, K.U.Leuven, Leuven, Belgium, 2001.

[3] `https://www.dropbox.com/s/cx2ex5o4n4q42ov/manual_v8.pdf?dl=0`

[4] `https://github.com/robclewley/pydstool`

[5] `https://pydstool.github.io/PyDSTool/FrontPage.html`

2.2 Tools overview

2.2.1 ODEs

1. XPP

   (a) Overview

   - Language: C
   - Interface: GUI only
   - Usage: ODEs, DDEs, SDEs, BVPs, difference equations, functional equations
   - License: GNU GPL V3

   (b) Notes The 'classic' simulation and continuation software. Still sees active use in a large range of nonlinear problems. Bifurcation (continuation) methods provided by AUTO and HomCont; probably possible to use AUTO by itself, but no one does because it would be very difficult (needs FORTRAN coding), and XPP provides a good interface to do it. Takes plain-text input files, with equations written out in text, as opposed to being defined by user-written functions like in eg. matlab. From [1], ... Over a dozen different solvers, covering stiff systems, integral equations, etc. Supports Poincare sections, nullcline plotting, flow fields, etc., so it's good for visualisation, as well as bifurcation analysis. Can produce animations in it (somehow?). Since it's so popular, there's a wealth of tutorials available for it. Somewhat outdated GUI, but it does the job perfectly adequately. No command line interface. Buggy, sometimes segfaults.

   (c) Tutorials Comprehensive tutorial provided by Ermentrout here: `http://www.math.pitt.edu/~bard/bardware/tut/start.html#toc`

2. **TODO** COCO
   (a) Overview
   (b) Notes
   (c) Tutorials

3. **TODO** MatCont
   (a) Overview
      - Language: MATLAB
      - Interface: GUI only, but $\text{CL}_{\text{MatCont}}$ exists as a command-line version
      - Usage: """"""TODO"""""""
      - License: Creative Commons Attribution-NonCommercial-ShareAlike 3.0 unported

   (b) Notes Also: $\text{CL}_{\text{MatCont}}$ (commandline interface), MatContM (MatCont for maps)
   (c) Tutorials

4. PyDSTool See the project overview for lots of nice interesting things to talk about
   (a) Overview
      - Language: Python3, with options for invoking C, Fortran
      - Interface: scripting only
      - Usage: ODEs, DAEs, discrete maps, and hybrid models thereof; some support for DDEs
      - License: BSD 3-clause

   (b) Notes Julia DS library is just PyDSTool in a julia wrapper. Provides a full set of tools for development, simulation, and analysis of dynamical system models. 'supports symbolic math, optimisation, phase plane analysis, continuation and bifurcation analysis, data analysis,' etc. (quoted from [5]). Easy to build into existing code. Can reuse bits and pieces (eg. continuation, or modelling) for building more complex software.
   (c) Tutorials Learn-by-example tutorials provided in the examples directory of the code repo [4], and fairly comprehensive documentation available on the website [5].

2.2.2 Others

1. DDE Biftool
   (a) Overview
      - Language: MATLAB
      - Interface: Scripting
      - Usage: DDEs, sd-DDEs
      - License: BSD 2-clause

(b) Notes DDE bifurcation analysis only. Described in detail at `http://twr.cs.kuleuven.be/research/software/delay/ddebiftool.shtml` . Full manual available at [2]. Designed for numerical bifurcation analysis of fixed points and periodic orbits, in constant-delay differential equations, and in state-dependent-delay differential equations. Uses orthogonal collocation (???) to continue steady states, periodic orbits. Doesn't provide automatic bifurcation detection, but instead tracks eigenvalue evolution, so that the user can determine bifurcation points. No simulation ability.

2. Knut

   (a) Overview
      - Language: C++
      - Interface: GUI, CLI
      - Usage: explicitly time-dependent-delay DDEs
      - License: GNU GPL

   (b) Notes
      i. Features: [Info taken verbatim from `https://rs1909.github.io/knut/`]:
         - Continuation of periodic orbits along a parameter
         - Floquet multiplier calculations
         - Automatic bifurcation detection
         - Continuation of some bifurcations in 2 parameters

      ii. Differences from DDE Biftool: [Info taken from `https://rs1909.github.io/knut/`]:
         - C++ makes it faster than MATLAB
         - Standalone software (no need to install matlab as well)
         - GUI-based, with plaintext input, so no need for any programming skills to use it
         - Only software to calculate quasi-periodic tori

   (c) Tutorials See reference manual [3] for how-to's

3. PDECONT

   (a) Overview
      - Language: C
      - Interface: combination of C and a config file. Matlab interface appears to exist, but no documentation for how to use it
      - Usage: PDE discretisations, large systems of ODEs
      - License: unspecified (open-source, and free for non-commerial use)

   (b) Notes Huge long documentation file exists, but that's just full of code implementations. Couldn't find any clear, straightforward tutorials for using it. Need to code in C and produce a big config file to use the software. Even then, I can't tell what the code is actually designed to do…

2.3 Tables

2.3.1 Point labels

Point	Label	Also known as
EP	Equilibrium	
LC	Limit cycle	
LP	Limit point	Fold bifurcation, saddle node bifurcation
H	Hopf	Andronov-Hopf bifurcation
LPC	Limit point of cycles	Fold / saddle node bifurcation of periodics
NS	Neimark-Sacker	Torus bifurcation
PD	Period doubling	Flip bifurcation
BP	Branch point	
CP	Cusp bifurcation	
BT	Bogdanov-Takens	
ZH	Zero-Hopf	Fold-Hopf, Saddle-node Hopf, Gavrilov-Guckenheimer
HH	Double Hopf	Hopf-Hopf bifurcation
GH	Generalised Hopf	Bautin
BPC	Branch point of cycles	
CPC	Cusp point of cycles	
CH	Chenciner	Generalised Neimark-Sacker bifurcation
LPNS	Fold-Neimark-Sacker	
PDNS	Flip-Neimark-Sacker	
LPPD	Fold-flip	
NSNS	Double Neimark-Sacker	
GPD	Generalised period doubling	

(Taken from the MATCONT Scholarpedia page)

2.3.2 TODO Types of curve

Curve label	Curve type	MATCONT	CoCo	AUTO	PyDSTool
EP-C	Equilibrium	y		y	y
LP-C	Limit point / fold	y		y	y
H-C1	Hopf (method 1)	y		y	y
H-C2	Hopf (method 2)	-		-	y
LC-C	Limit cycle curve (family of POs)	y		y	y
	Limit point of cycles	y		?	?
	Period doubling	y		y	**
	Neimark-Sacker	y		y	**
	Homoclinic to saddle	y		y	n
	Homoclinic to saddle-node	y		y	n
*	Branch point	y			
*	Branch point of cycles	y			
*	ConnectionSaddle	y			
*	ConnectionSaddleNode	y			
*	HomotopySaddle	y			
*	HomotopySaddleNode	y			
*	ConnectionHet	y			
*	HomotopyHet	y			
*	Heteroclinic	y			

* What do thes mean? Are they actually a bifurcation curve type? ** PyDSTool seems to have methods to compute these for fixed points of maps; does that mean they're a maps-only type of curve? Note that it lacks documentation and tests/examples about these methods, so maybe they're not implemented? ? indicates that there doesn't appear to be a native way of doing this, however it's possible that there's ways to do it (eg. AUTO97 apparently let's us track LPCs, and PyDSTool let's us define custom curves to follow, so one could possibly construct a customised continuation regime to track limit points of cycles)

2.3.3 TODO Types of point

Point type	Codim	MATCONT	CoCo	XPP	PyDSTool
LP	1	y		y	y
H	1	y		y	y
LPC	1	y			y
NS	1	y			y
Torus bif				y	
PD	1	y		y	y
BP	2	y		y	y
CP	2	y			y
BT	2	y			y
ZH	2	y			y
HH	2	y			y
GH	2	y			y
BPC	2	y			n
CPC	2	y			n
CH	2	y			n
LPNS	2	y			n
PDNS	2	y			n
LPPD	2	y			n
NSNS	2	y			n
GPD	2	y			n

* Are branch points just 'there's a bifurcation here but we don't know what type specifically'? In that case, any bifurcation that occurs, but isn't one of the labelled ones, would still be detected as a BP. Also see the MATCONT 'objects related to homoclinics to equilibria' table, and resonances, for additional points it can detect

2.3.4 TODO Available numerical methods

Method	MATCONT	CoCo	XPP	PyDSTool

2.3.5 TODO Types of system they can simulate

System	MATCONT	CoCo	XPP	PyDSTool
ODE			y	y
PDE (discretized)			y	n
DDE			y	limited
SDE			y	limited
DAE			y	y
BVP			y	n
Maps			y	y
Hybrid			basic (apparently)	y
Integral			y	n
Difference			y	y
Functional			y	n

While XPP is capable of simulating all the noted systems, I don't know if that is literally just XPP simulating them, or also that AUTO is able to run continuations with them

Aren't difference equations the same as maps?

2.3.6 TODO Degree of manual fiddling / parameter tuning

2.3.7 To code or not code?

MATCONT	XPP	PyDSTool	CoCo
No coding necessary	No coding necessary	Coding required (matlab)	Coding required (matlab)

2.3.8 License

MATCONT	XPP	PyDSTool	CoCo
Creative commons, but requires a matlab license	GNU GPL v3	BSD 3 clause	None specified; matlab licens

There might be the option of running matcont or CoCo in GNU Octave, meaning no matlab license is required, but this is not a given.

2.3.9 TODO Crashing and instability / ease of use

2.3.10 TODO Other stuff

Thing	MATCONT	CoCo	XPP	PyDSTool
Toolboxes	biomechanical, compneuro, systems biology			
Auto C code generation	Yes, for ODE/ DAE / map simulations			
Bounds safety	Yes, can preserve eg. non-negativity			
Index-free system	Yes, making for clear syntax			
Extensible	Yes, can easily build on the code and expand it			
Heirarchical model composition	Yes			
Events detection	Yes			
Symbolic manipulation	Yes			
Memory management utilities	Yes, inc. LaTeX markup export, smbl conversion			
Parameter estimation / fitting	Yes, toolboxes for that			

2.3.11 TODO PyDSTool vs others

PyDSTool	XPP	MATCONT
Arbitrarily large systems	No heirarchical composition-based modelling	
Wider range of DE RHS, but no stochastics	Supports stochastic RHS	
SUpports long names	9 character max. for names	
Scriptable	Not scriptable	
Can embed simulations in other environments	Can only use as a standalone box	
Limited DDE support	Supports general DDEs	
Fewer integrators than XPP	Supports more ODE integrators than PyDSTool	
No BVP solver	Has a BVP solver	
Slower than XPP, as fast as MATCONT	Written in C / fortran. Fast!	Slower than X
Closer integration with the programming env	Hard to interface with other programming	Harder to int