PRESENTATION NOTES, DON'T DISPLAY THESE!

Mark Blyth

Last meeting

Last time:

- The challenges of working with spiking signals
  - Lots of high-frequency energy
  - Typically noisy, both from stochastics and from measurement errors
  - Lots of high-frequency components mean a LP filter will remove the signal, as well as noise, so we can't naively clean up the signal
  - Also means using a truncated Fourier discretisation will be infeasible, since it'll have far too many components to effectively discretise
- Surrogate methods to overcome these:
  - Use a regression model in place of the real data
  - Perform desired analysis on this instead
  - 'Desired analysis' will be explained more later, in a CBC context
  - A well-chosen surrogate will filter out all the noise, without losing any signal

Surrogates point 1

- Given $y_i = f(x_i) + \varepsilon$, estimate $f(x)$
  - We assume there's a 'true' underlying signal f(x)
  - This true signal is what the neuron is actually doing, eg. what the membrane potential actually is at the patch clamp location
  - We don't have access to f(x); instead, we get a time-series y_i, of noise-corrupted samples
  - These noise-corrupted samples contain both the actual signal at the given sample time, plus some nuissance variable ε from errors in measurement
  - We wish to recover f(x) from these samples, as that's the noise-free, true signal that we're interested in
    - Simply LP filtering would only remove the HF components of f(x) and ε; instead, we wish to separate the two out into noise and signal
  - We can use statistical methods to infer f(x) and ε
  - This gives us a clean, noise-free surrogate to perform all the analysis on
  - Surrogate: we use f(x) in place of the real data

Surrogates point 2

- Splines and Gaussian processes are good methods for estimation
  - GPR: mathematically elegant, rigorous method
    - Assume normally distributed ε
    - Point estimate: y ~ N(f(x), ε)
    - Whole function estimate: f(x) ~ GP(mu, Sigma)
    - Whole function estimate is actually a Gaussian distribution over functions
    - The whole function estimate is just a generalisation of the point estimate
    - For a sensible prior, we can then use Bayes to estimate the posterior distribution on f(x)
    - Statistically optimal estimator
    - Downside: for finite data, results are only as good as the priors we use; coming up with good priors is hard
  - Splines: simple, effective, less elegant
    - Split f(x) into intervals, and asume f(x) is locally polynomial on any given interval
    - Enforce C^2 smoothness over polynomial sections
    - Polynomials then join up at the edges of each interval; these joinings are called knot points
    - Remaining free parameters are chosen to maximise goodness-of-fit
    - No need to define priors, so it's easier to use on data where choice of priors becomes difficult
    - Knot points are difficult to choose; use Bayesian inference to form a posterior distribution over knots

Developments since last time

IMAGE.

- Surrogates tested and working
  - GPR works on both real and synthetic data, in cases where the data are sufficiently stationary
  - Free knot splines works always, so use it in cases where the data aren't sufficiently stationary
- Image taken from recent abstract
  - Bayesian free-knot splines
  - Works well – we can extract the underlying signal near perfectly, even given very noisy observations
  - Three changepoints per period: at the start, top, and end of a spike; here, the signal rapidly changes from slow to fast behaviour
  - These changepoints are the hardest bits to model, and therefore the surrogates are least accurate here (the time between spikes shows a slow, gentle change that's easy to model accurately)
  - Zooming in on one of the changepoints, we see that the surrogate recreates the latent signal nearly exactly; even at the most difficult-to-fit part of the signal, we still get excellent results

Developments since last time

- Novel discretisations
  - Surrogates only give us a noise filter
  - For some CBC implementations, this is sufficient
  - In CBC cases where we have to do Newton iterations, this isn't useful; we instead need a low-dimensional discretisation
  - We can apply the surrogates ideas to creating discretisations
- In-silico CBC
  - Best way to demonstrate that these methods work, are valuable

Discretisations

- Discretisation takes a function, projects it onto a set of basis functions
- Coefficients and basis functions are sufficient to represent the signal
- Lots of possible choices for basis functions
  - C^{infinity} signals can be represented exactly with monomial basis functions (taylor expansion)
  - Periodic signals can be represented exactly with trig basis functions (Fourier series)
  - These are bad choices for neuron CBC – require lots of coefficients to describe the spiking signals
- We've already met splines; turns out we can define a set of basis functions for splines
  - Can therefore express any spline curve in the above form
  - This means we can discretise with splines too!
  - Splines are a good choice: they provide a nice simple, intuitive model, and don't require many basis functions to get a good approximation

Splines discretisation

- Fit a set of basis functions to initial signal $f_0(x)$
  - Choose a set of knots xi, such that the splines basis $b_i(x)$ that we construct from knots xi is able to fit the initial signal $f_0(x)$ as accurately as possible, in the least squares sense
  - This is actually hard to do – open research problem
  - Elegant approach: find a Bayesian posterior over $\xi|data$. Downside: is slow and complicated; need to do MCMC to approximate intractable integral
  - Simple approach 1: put a knot at every datapoint then penalise functional of second derivative, to enforce smoothness. Downside: we end up with huge numbers of knots.
  - Simple approach 2: keep adding knots until we reach satisfactory results; downside: lower quality fit, no guarantee of low-dimensionality
- My approach: choose the number of knots; numerically optimise knot positions; start from random initial knots; avoid local minima by repeating this lots
  - Downside: need to repeat lots to find global minimum
  - Need to choose the number of knots a priori; algo doesn't work it out for us
  - Upside: quick and easy approach to finding a good set of knots; easiest way to get low-dimensional knot set

Splines demo

- Splines discretisation works well
- This example uses just 8 knot points
  - Higher than an 8d discretisation, as we need to add exterior knots so that the basis splines have support across the range of the data
- Reconstructs the latent signal near-perfectly

Splines vs Fourier

Also shown: Fourier

- Visually, splines fits better than Fourier
- Fourier is harder to fit
- Too few harmonics and the series can’t fit the data
- Too many harmonics and the series starts fitting the noise as well as the data
- Not really any sweet spot; no point where the series fits the signal, but averages out the noise
- This is the usage case for surrogates – when we have noisy data, but still want to use Fourier with it!

Goodness-of-fit

This shows the goodness-of-fit of a splines model with given number of knots, and Fourier series with given number of harmonics

- No noise, Fitzhugh Nagumo
- Splines error decays more rapidly than Fourier error
- Effects become even more dramatic for more neuron-like signals
- Note though this is the goodness-of-fit of a splines, fourier model on a single signal; doesn't determine how well the splines model generalises to discretising unseen signals, ie. only shows how well the spline model fits a signal to which its basis functions were fitted; using the same basis functions on a signal from a different parameter value might get different goodness-of-fit. Fourier won't have this issue since it uses trig basis all the time

Method usage cases

- Harmonically forced:
  - When we have a harmonically forced system, we can have a harmonically oscillating control action, and treat the control action as the forcing term
  - In this setup, we can efficiently iterate on the Fourier harmonics, to drive the higher-order harmonics of the control action to zero
  - This necessitates a Fourier projection. No need for a novel discretisation, but we could possibly improve the Fourier discretisation by using a surrogate to first filter off the noise
- Non-harmonically forced:
  - If system is unforced, we apply parameter and control action separately, and need the control action to be zero
  - We can use Newton iterations to solve for the noninvasive control action
  - Since we're doing Newton iterations, we need to work with a low dimensional system, otherwise it'll be impractically slow
  - To have a low-dimensional system, we use a novel discretisation, eg. splines

In-silico CBC

Current work: implementing an in-silico CBC simulation

- Best way to test if the surrogates, discretisations work with CBC is to try using them with CBC!

CBC method POINTS 1, 2

- Use PD control
  - Easy, model-free control method
  - Gets good results with a method we could easily use in experiments too
  - Fit control parameters with brute force
  - Easy to simulate, minimal effort in controller design
- As per standard numerical continuation, do a change in variables so that time is in [0,1], and treat period as an extra continuation variable
  - Not necessary with Fourier discretisation
  - Splines knots are like finite-differences or collocation mesh points
  - Time rescaling is necessary with mesh-based methods, as changing the period would effectively move the mesh points relative to the signal

CBC method POINTS 3, 4

- Non-adaptive mesh
  - Fit knots at the start, keep them in the same position throughout
  - Adaptive mesh would mean re-fitting the knots after a prediction-correction step
  - In terms of code, this is minimal extra effort, but would add a slight fitting overhead
  - Only using non-adaptive mesh because I'm interested to see how well it works
- Use Newton iterations to solve for discretised control target = discretised system output
  - Nothing fancy, just simple, slow Newton with finite differences; able to do this in-silico, but would need more rigorous treatment for experiments
  - Sensible to start off with easy root finding, and develop something fancy (Broyden) later
  - Splines method adds exterior knots, and some of the coefficients are always zero, so they can be removed from the discretisation to speed up the finite-differences Jacobian step; I'm currently being lazy and not doing this

Discretisors

- Instantiate desired discretisor type with its relevant parameter
  - Fourier: $n_{harmonics}$
  - Splines: knot locations
  - Regardless of discretisor type, can then call discretisor discretise, discretisor undiscretise
- Simple, standard interface to discretisation routines
  - Able to swap between Fourier, Splines with zero effort
  - Allows direct comparison between discretisation methods
  - Could easily implement any other discretisation (eg. wavelets) using the same interface
- Lightly tested:
  - The code runs and produces sensible outputs
  - Haven't tested its ability to generalise to new signals
  - Ie. don't know how well basis funcs fitted to $f_0(x)$ will work for discretising $f_1(x)$

Controllers

- Can design controllers with a standard interface too
  - Set the controller type, control target, gains, and the control matrices
  - The controller object handles the rest
- Similarly, can design systems with a standard interface
  - Specify a function that gives the ODE RHS; a list of ODE parameters; the controller
  - Can then run the controlled model for any choice of time range, ICs, pars
  - Subsequent runs optionally start with ICs given by final state of last run, much like a real system

The point of all these standardised interfaces is that it becomes really easy to swap everything out; eg. apply to different models, different control strategies, different discretisors

Can then run a CBC experiment in less than 10 lines of code; easy to apply, reapply, experiment with

Control

IMAGE

FH system with sine target; looks very reasonable

- Lightly tested: code runs, results look very reasonable
  - Seems like a sensible output
- Can easily write out the RHS of a PD-controlled FH system w/ sine target; can compare this explicit RHS to the code-generated system, to make sure the code isn't doing anything funny
  - Haven't done this yet

Continuation

- Runs a psuedo-arclength secant-predictor Newton-corrector CBC
- Code is written now
- Requires a ‘system’: this is just a controlled model [from previous code], with its arguments binded
- ‘system’ interface will be easy to write, but haven’t got round to this yet

Simulation summary

- Results handling is just something to take the set of natural periodic orbits, apply some measure (eg. amplitude), then plot them on a bifurcation diagram.
- Should be working and tested within a week

Open questions

- Will splines discretisation work?
  - If splines can only model the signal to which the knots were fitted, they won't work for CBC
  - My guess is they will work
- Stationary or adaptive mesh?
  - If splines basis aren't good at generalising, can re-fit knots at each step, much like an adaptive mesh, which would hopefully fix problems
- Efficient solving methods
  - Remove zero-coefficients from discretisation
  - Broyden Jacobian update?
  - Newton-Picard iterations? Ludovic's suggestion of Newton-iterating on unstable coefficients, fixed-point iterating on stable coefficients; reduces the size of the Jacobian / finite differences step
- Can we interface the code with Simulink?
  - Ludovic has a simulink model that would be fun to play with; haven't looked at it yet since I've been testing the CBC codes; would be interesting to try to call the finished code from MATLAB, in which case we might be able to interface the two

Next steps

- Continuation tutorial paper
  - Haven't touched it recently
  - Making slower progress since I'm trying to get the stuff for this done before the paper deadline
- NODYCON abstract for this: submitted, accepted
- Conference paper for this: will start on that once the CBC simulation is sorted