Plan of action, brute force testing

Mark Blyth

Things I did

- Collocation
  - Coded up numerical continuation with collocation
  - Redesigned collocation equations for BSplines
  - Coded up BSpline numerical continuation
- Planning
  - Reviewed all the work on discretisations so far, for inspiration
  - Drew up a loose plan for the year
- Testing
  - Systematically tested lots of combinations of control gain, solver, discretisation
  - Discovered probable cause of spline failure

Overall PhD goal

Goal: use CBC to classify bursting neurons by slow/fast analysis

- There's lots of burster types, classified by their fast-subsystem singularity and codimension
- Krasi showed that a cubic Lienard model describes many burster fast-subsystems
- Possible way to classify...
  - Map out the critical manifold *[hard!]*
  - Produce a bifurcation diagram of the fast subsystem
  - Fit the cubic Lienard model from the results
  - Use that to classify the neuron

Combines CBC, multiple timescale analysis, and system identification in a *[hopefully]* biologically useful way

Overall PhD goal

- Key challenge: separating out the fast and slow subsystems
  - With models, we can separate equations into fast and slow components
  - This reveals critical manifold and its bifurcations
  - No obvious way of doing this experimentally, would require other approaches
  - If this can be done it would potentially be a big result

Questions for Krasi:

- This is mathematically interesting, but is it biologically interesting / useful?
  - Where has burster classification been useful to biologists?
- How complex are *real* burster dynamics?
  - There's a lot of complicated multiple-timescale dynamics (torus canards, MMOs) available in neuron models
  - Are these dynamics just mathematical toys, or are they seen in live cells too?

High-level year plan

- Paper 1: get BSpline discretisation working; target: done by March
- Paper 2: a set of extensions that didn't make it to paper 1; target: done by May
  - Try some other discretisation methods, since most of the work is done for them
  - Create a recipe book of which discretisation to use when
  - Demonstrate the discretisations on any existing CBC rig?
- Then, investigate multiple-timescale analysis, nonlinear model identification, and active learning; rest of the year?

Plan: immediate

- Test results suggest that adaptive stepsizes might be the secret to splines success
- TODO: implement and test adaptive stepsizes

Plan: month

Assuming stepsizes solves all the problems...

- Try wavelets in place of BSplines
  - *Should* be quick and easy
- Look into adaptive...
  - Knots for BSplines
  - Number of knots?
- Demonstrate results on some prototypical systems and write up into a paper

Aim to start writing up by the end of January

Plan: exensions

- Collocation codes could be used with CBC easily
- Maybe try some novel CBC approaches
  - Collocation discretisation
  - LSQ collocation
  - Bayesian opimization as a surrogate solver

- Solves similar problems to BSpline/Wavelets paper so its less useful
- ...but lots of the work for this is done, so it could be easy extra paper
- These ideas hopefully won't take much time to test
  - Test alongside writing up previous stuff

Collocation and BayesOpt

- If we have more collocation points than coefficients, we can fit coefficients in a LSQ sense
  - More numerical robustness: guarantees CBC solution exists, even if discretisation is inexact
  - More noise-robustness: noise is averaged off by LSQ
- Alternative: use Bayesian optimization to minimise $\|\text{target}(t) - \text{output}(t)\|$
  - Bayesian optimization is basically a surrogate method
  - Accuracy: gets results fast even with higher-dimensional discretisations
  - Robustness: noise is explicitly modelled, and averaged off

Might make a nice paper or conference paper

Another possible extension

Lots of novel discretisations available

- Wavelets
- BSplines
- Collocation
- BayesOpt

Test them out with CBC…

- In the presence of noise
- In experiments, on any available, existing CBC rigs

Possible outcomes:

- A ‘recipe book’ of which discretisor to choose when
- An adaptive method that automatically chooses the best discretisation at each predictor/corrector step

Plan: beyond discretisation

Look into slow/fast systems, and nonlinear model identification

- Use a CBC bifurcation diagram to propose system models
- Link CBC directly into the model generation
  - Active learning procedure
  - Model identification routine decides what data will be most informative at each step
  - CBC is used to obtain that information

Hopefully start this in spring, after discretisors are done

- Start some reading now!

Brute force testing

- Collocation was motivated by some numerical quirks with BSplines
- Decided to systematically test control gain, solver, discretisation, to see exactly what quirks arise when
  - Could also have varied discretisation size, but that's never seemed to have much of an impact with Duffing data
- Shows what the quirks are and where they arise
  - Conclusion: use adaptive stepsizes

Controllers: what and why?

Key takeaway: fails at basically the same place, in the same way, regardless of K_p, suggesting that the issues aren't down to the controller or controllability

Can't solve the IO-map if we can't control the system, so K_p is worth testing

- I'd expect low K_p to not work: too small means we can't control the system; *agrees with results*
- I'd expect large K_p to not work: too big means negligable proportional error, negligable gradient in solution-neighbourhood, so hard to solve accurately and to spot when control is noninvasive; *agrees with results*
- I'd expect a wide range of middle-ground K_p, as tested, to work; *agrees with results*

Controllers: what and why?

Results conform to expectations, suggesting no need for fancier control strategies here

Interesting project for someone: what's the best control strategy?

- Strong enough control to steer the system properly
- Gentle enough control for the continuation equations to remain numerically solvable
- Can we always find a K_p sweetspot?

Solvers: what and why?

Key takeaway: no concerning differences between different solvers, suggesting they're not at fault

- Both solvers should get basically the same solution
  - Noninvasive control is noninvasive control, regardless of how we find it
  - Might expect to see a little difference between SciPy and DIY solver; DIY solver uses fixed finite-differences stepsizes; SciPy solver will be more clever, more accurate
- Both solvers struggle on second stable branch with BSplines
  - SciPy solvers give up at same place as Newton accepts incorrect solutions
  - It looks like they're failing in the same places; SciPy knows its failing, Newton doesn't
  - Reassuring: suggests success or failure is reasonably solver-indepenent
- Matches previous results; can't always get a spline solution
  - Previous suggestion: maybe a solution doesn't exist
  - New suggestion: maybe this a numerical, rather than existence-and-uniqueness, issue?

Discretisors: what and why?

- Ideally we would expect near-identical results between Fourier and splines
  - Noninvasive control is noninvasive control, regardless of whether it's expressed with splines or Fourier
- Generally this seems to be true
  - I expect the differences in discretisor actually arise from the different stepsize requirements
- Comparing discretisors is slightly tricky
  - Ideally we want to keep everything the same except the discretisor
  - Different stepsizes will work best for different discretisations, so it's hard to do a like-for-like comparison
  - This reveals something interesting!

Stepsizes: what and why?

- In the current code, we use a fixed stepsize
- I'd tried varying stepsizes, with no success
    - Perhaps a small stepsize is required for success in some places, and a large stepsize in others
    - Perhaps choosing big steps will fail on one part of the curve, small steps will fail elsewhere
    - Never saw success when varying stepsizes, maybe because there's no single stepsize that works for the whole curve

If true, adaptive stepsizes will fix everything!

Summary

- Success is reasonably independed of control gain
  - K_p should be not too big and not too small
- Success is reasonably independent of solvers
  - Main difference is that SciPy will be more accurate when it works, and it knows when it's failed
- Success is reasonably independent of discretisation
  - Spline and Fourier success is down to stepsizes, rather than discretisations
- Success *is* dependent on stepsize!
  - I'd tested stepsize in the past, but with no success
  - Perhaps then, small steps fail in some places, large steps fail in others
  - Adaptive stepsizes might fix everything!

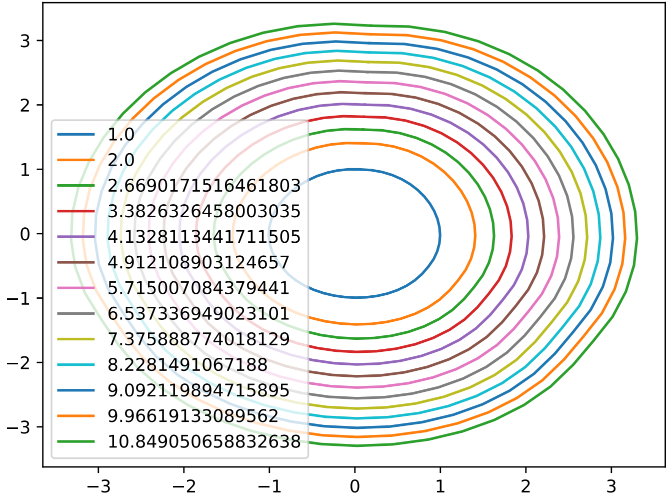
Orthogonal collocation

Coded up using the standard collocation method:

- Express periodic orbit as a BVP, and rescale to unit interval
- Split interval up into mesh, and model solution as a polynomial within each mesh segment
- Choose each segment's collocation points as (scaled) zeros of Legendre polynomials
- Find the polynomial coefficients that
  - exactly solve the BVP at the collocation points;
  - result in continuity between intervals;
  - result in periodicity;
  - also satisfy the phase constraint.

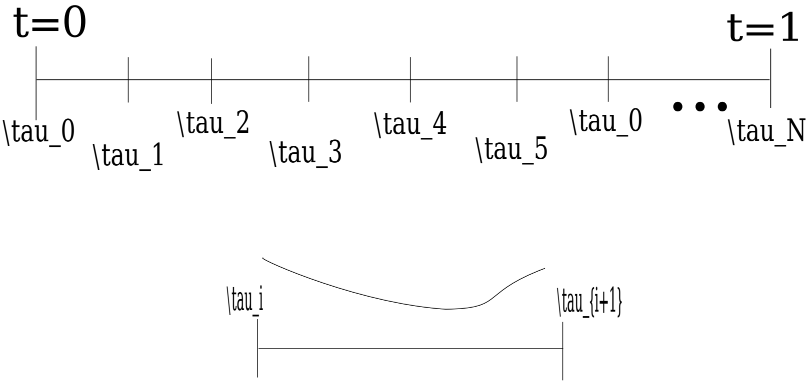
Results

It works! Continuing periodic orbits from a Hopf normal form:



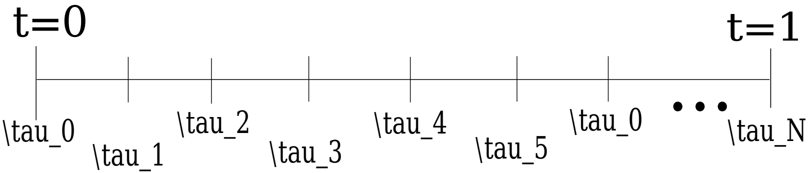
Collocation meshs

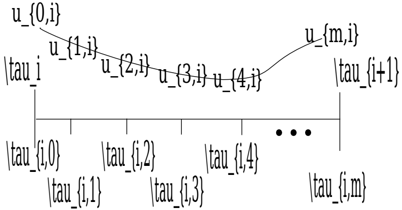
Standard continuation uses Lagrange polynomials in each subinterval



Collocation meshs

Lagrange coefficients are the value of the polynomial at a set of submesh-points $\tau_{i,j}$





BSpline collocation

BSPline knots form their own mesh, so we could either

- Use a set of BSplines within each mesh subinterval, so that the knots define $\tau_{i,j}$...
  - Solution is made of curve sections, with each curve made of polynomial sections
- ...or let the BSplines define the mesh τ_i and use a single set over the entire interval
  - Solution approximation is piecewise-polynomial between meshpoints, much like with standard continuation

I chose the latter (also done in a BSpline BVP paper)

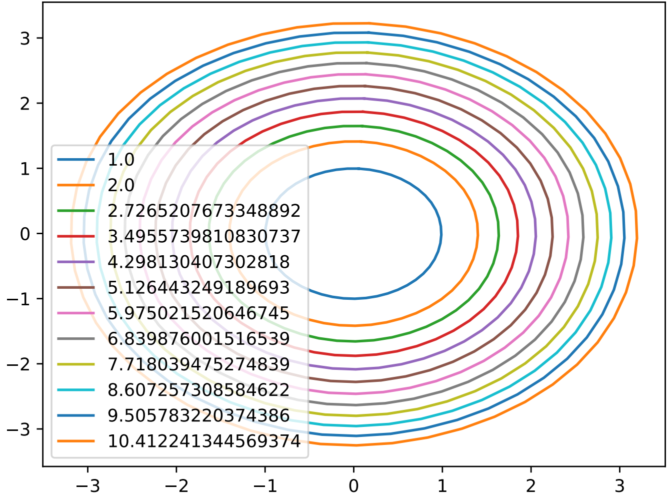
- Nearly identical to standard continuation, only we enforce a maximally smooth solution
- I use a periodic BSpline curve
  - Removes the need for periodicity equations
- Spline curves are maximally smooth
  - Removes the need for the Nn continuity equations

BSpline collocation

- Find periodic orbits by solving a periodic BVP
- Rescale BVP to unit interval
- Place BSpline knots evenly across the interval
- Add exterior knots to create a periodic BSpline curve
- Find the BSpline coefficients that
  - exactly solve the BVP at the collocation points;
  - satisfy the phase constraint.
- Choose collocation points as (scaled) zeros of Legendre polynomials

Results

It works! Just as easy to use as standard collocation

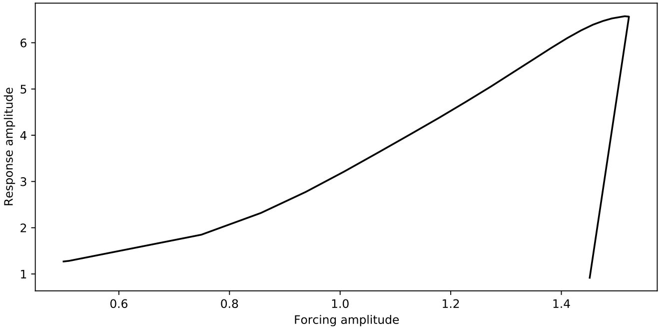


Next steps

Current TODO: implement and test adaptive stepsize continuation

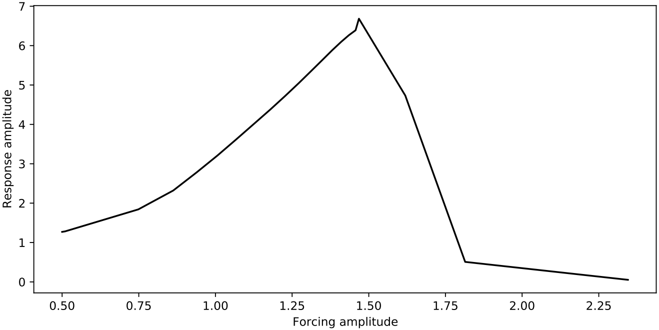
Also, any idea how I get the funding for NODYCON registration?

Splines, SciPy solver, $K_p = 0.25$



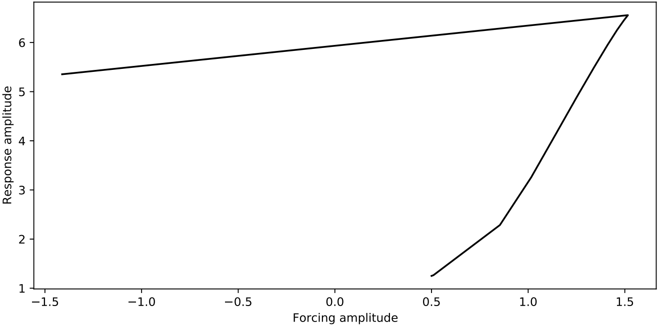
Can't control UPO

Splines, DIY Newton solver, $k_p = 0.25$



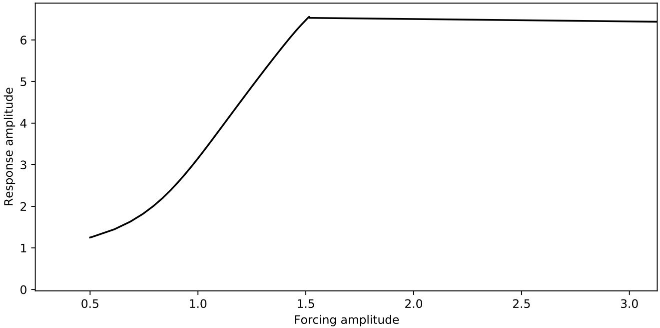
Bad convergence tolerance means non-solutions are accepted

Fourier, SciPy solver, stepsize=1, $K_p = 0.25$



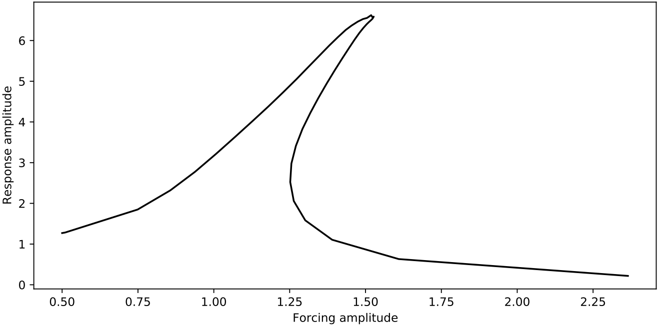
Can’t converge even to first point on curve

Fourier, SciPy solver, stepsize=0.2, $K_p = 0.25$



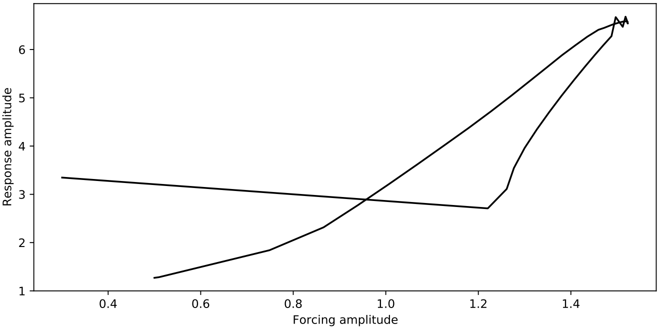
Can't control UPO

Splines, SciPy solver, $K_p = 0.5$



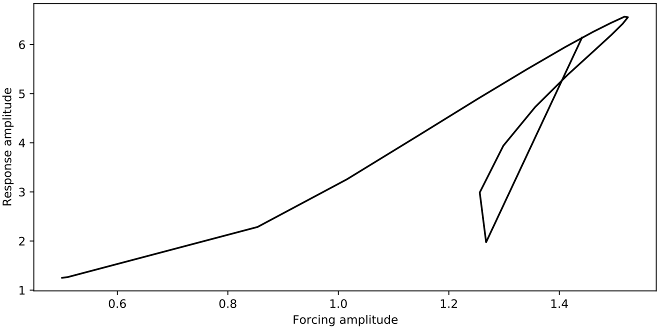
Works, but takes a huge final step and misses off a lot of the SPO

Splines, DIY Newton solver, $k_p = 0.5$



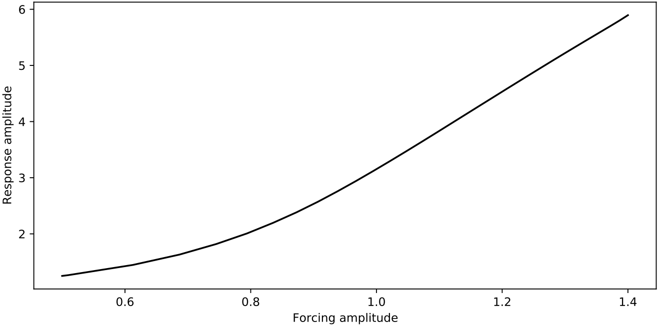
Doesn’t converge properly

Fourier, SciPy solver, stepsize=1, $K_p = 0.5$



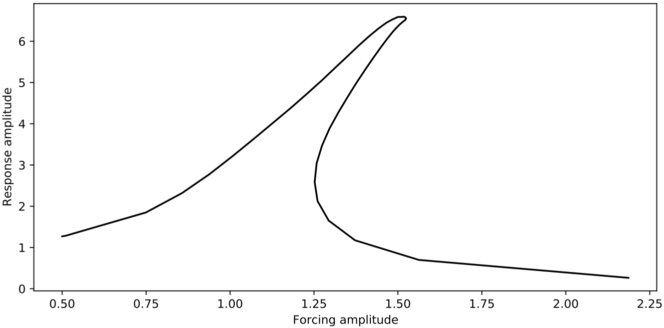
Huge steps; convergence fails part way along the second SPO branch

Fourier, SciPy solver, stepsize=0.2, $K_p = 0.5$



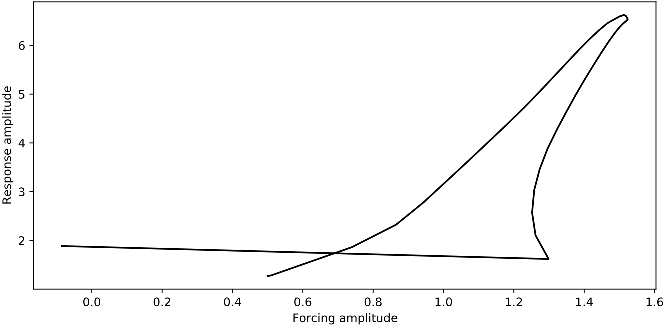
Fails to control UPO

Splines, SciPy solver, $K_p = 1$



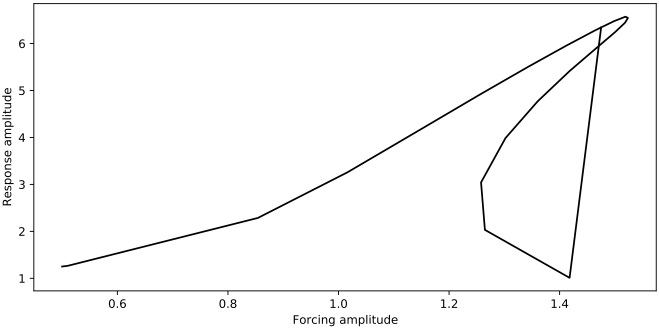
Works, but takes a huge final step and misses off a lot of the SPO

Splines, DIY Newton solver, $k_p = 1$



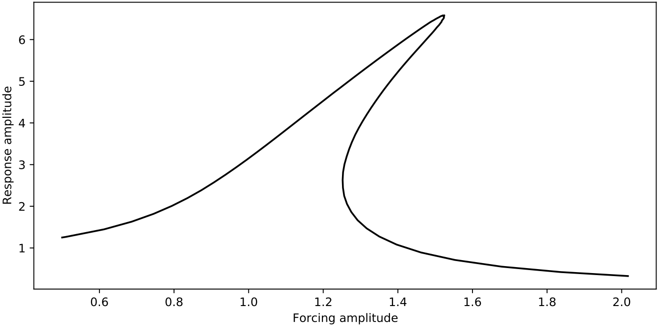
Doesn't converge properly

Fourier, SciPy solver, stepsize=1, $K_p = 1$



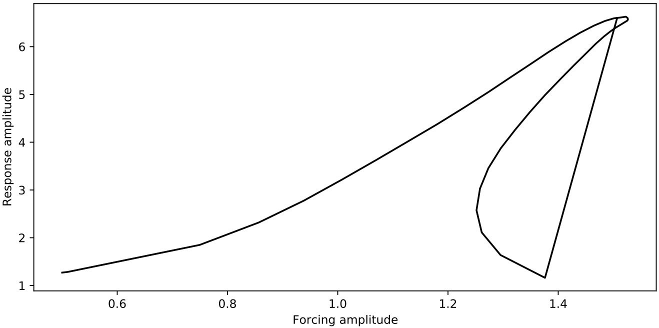
Fails to converge properly

Fourier, SciPy solver, stepsize=0.2 $K_p = 1$



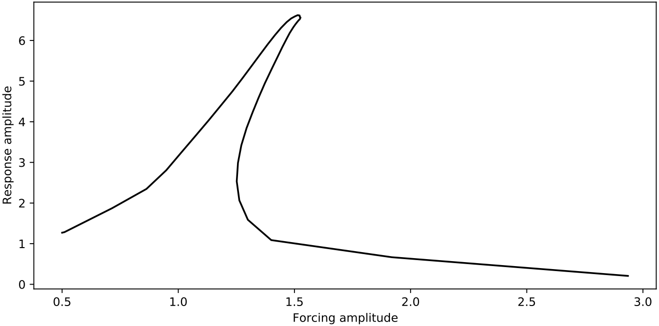
A perfect success!

Splines, SciPy solver, $K_p = 1.25$



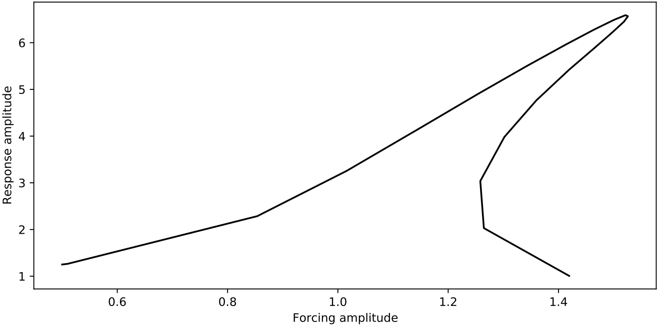
Fails to converge to next point

Splines, DIY Newton solver, $k_p = 1.25$



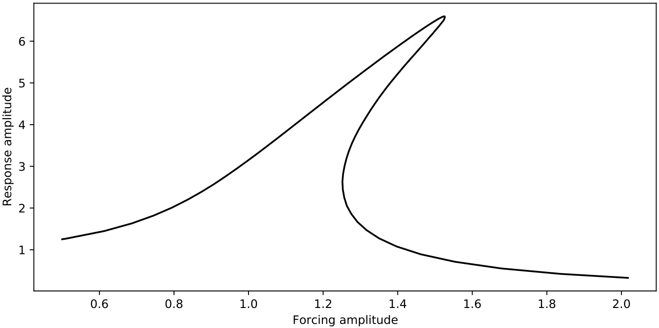
Converges, albeit with huge, inaccurate step from the same point SciPy failed

Fourier, SciPy solver, stepsize=1, $K_p = 1.25$



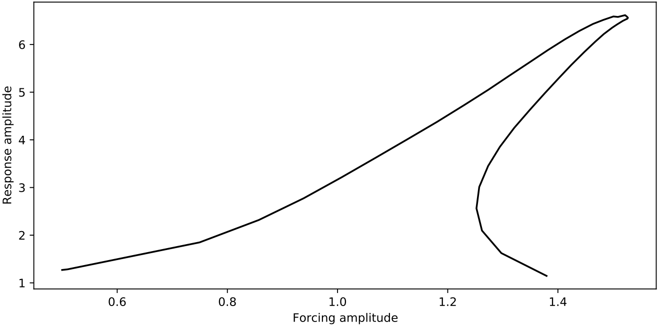
Continuation finishes early due to convergence failure

Fourier, SciPy solver, stepsiez=0.2, $K_p = 1.25$



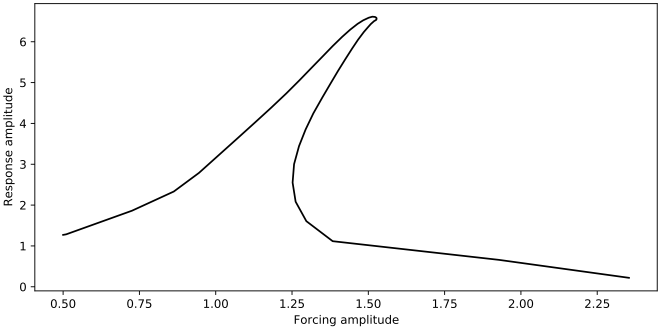
Perfect success

Splines, SciPy solver, $K_p = 1.35$



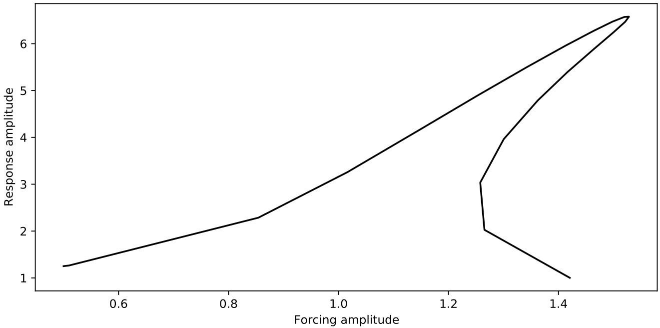
Continuation terminates early due to convergence failure

Splines, DIY Newton solver, $k_p = 1.35$



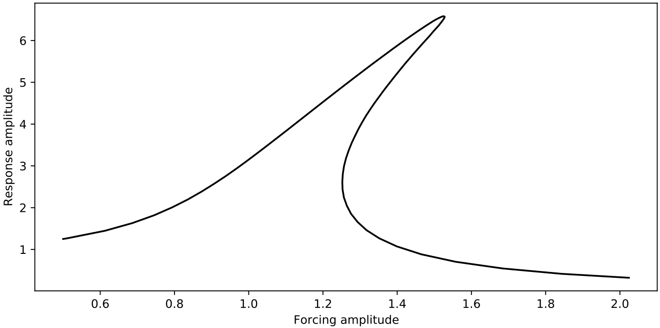
Converges, albeit with huge, inaccurate step from the same point SciPy failed

Fourier, SciPy solver, stepsize=1, $K_p = 1.35$



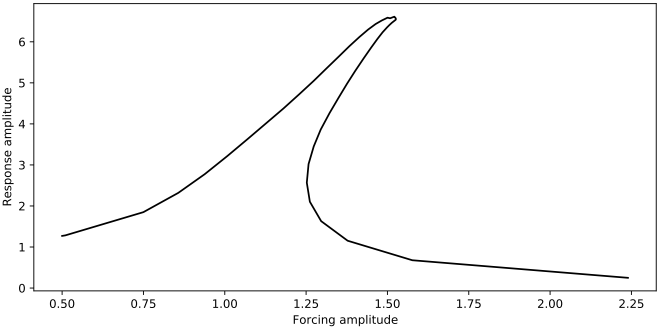
Continuation finishes early due to convergence failure

Fourier, SciPy solver, stepsize=0.2, $K_p = 1.35$



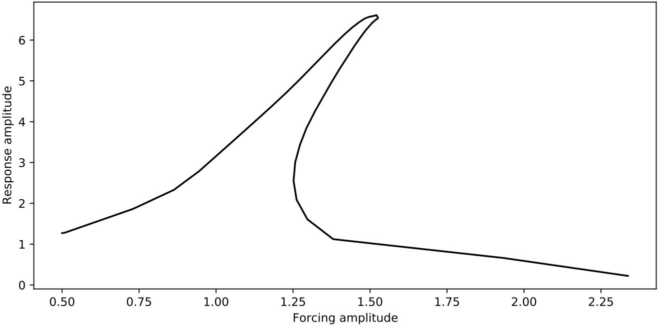
Success

Splines, SciPy solver, $K_p = 1.4$



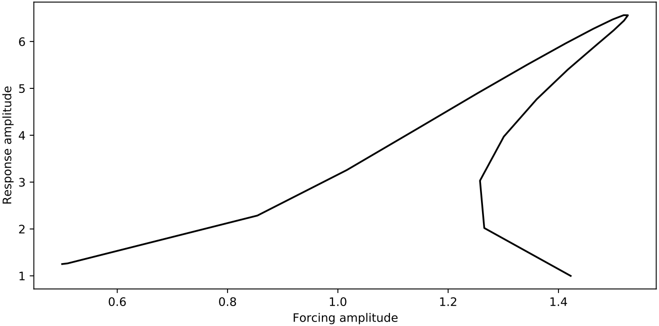
Works, but takes a huge final step and misses off a lot of the SPO

Splines, DIY Newton solver, $k_p = 1.4$



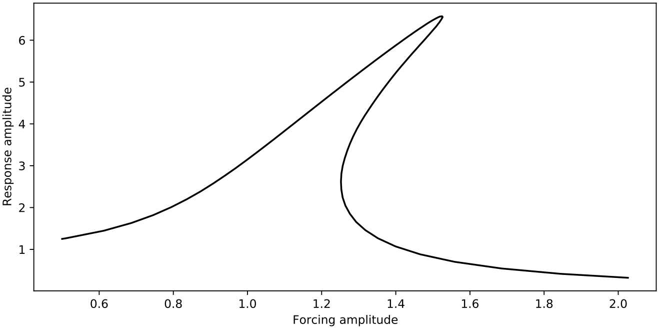
Skips most of the points along second SPO branch

Fourier, SciPy solver, stepsize=1, $K_p = 1.4$



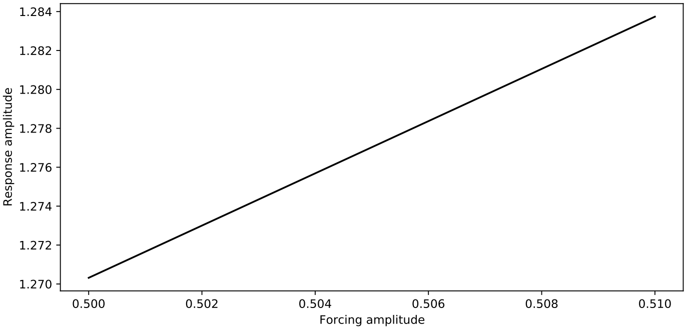
Continuation finishes early due to convergence failure

Fourier, SciPy solver, stepsize=0.2, $K_p = 1.4$



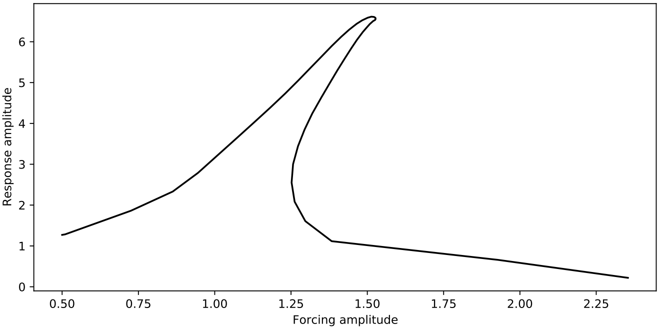
Success

Splines, SciPy solver, $K_p = 1.45$



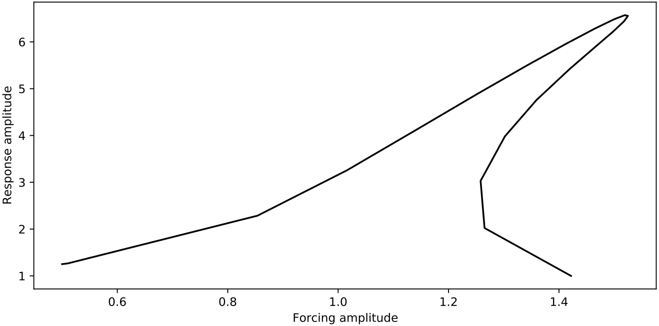
Fails to converge to even the first point

Splines, DIY Newton solver, $k_p = 1.45$



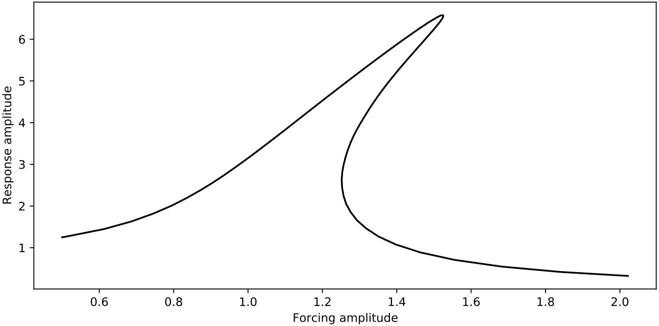
Converges, albeit with huge, inaccurate step

Fourier, SciPy solver, stepsize=1, $K_p = 1.45$



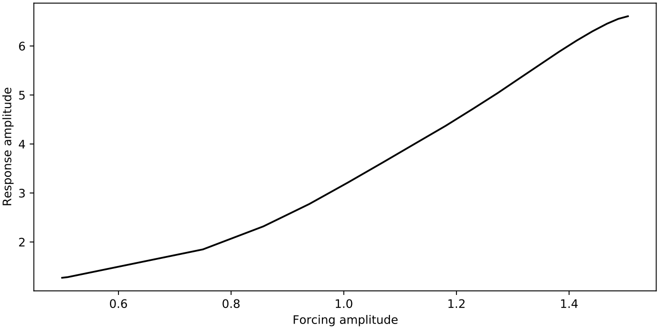
Continuation finishes early due to convergence failure

Fourier, SciPy solver, stepsize=0.2, $K_p = 1.45$



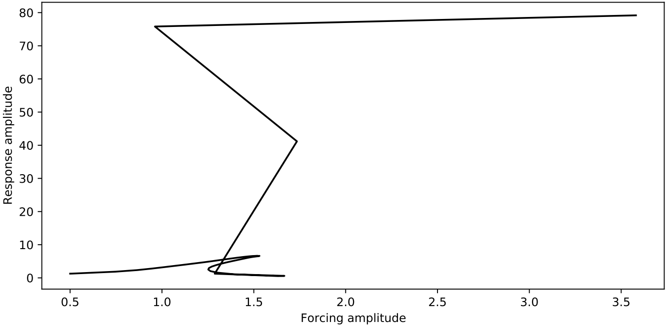
Success

Splines, SciPy solver, $K_p = 2$



Fails to control UPO

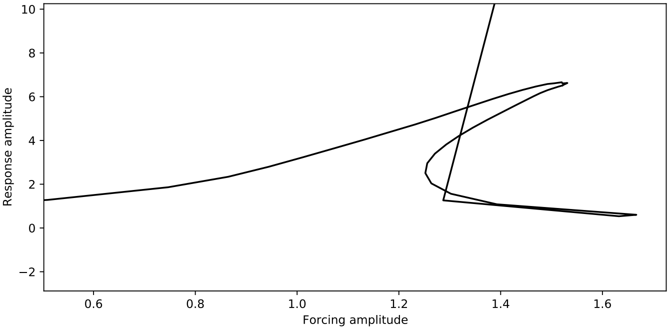
Splines, DIY Newton solver, $k_p = 2$



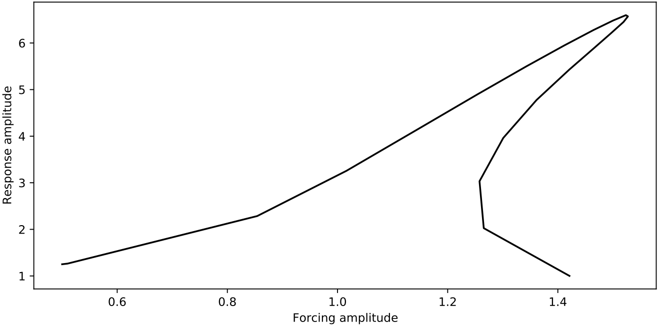
Fails at the usual place, then accepts non-solutions as the size of their Newton-update step is small

Splines, DIY Newton solver, $k_p = 2$

Zoomed in a bit

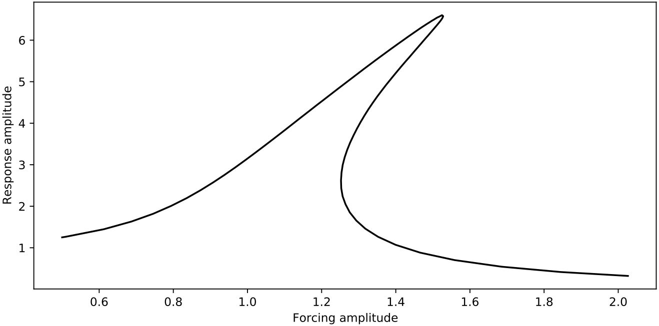


Fourier, SciPy solver, stepsize=1, $K_p = 2$



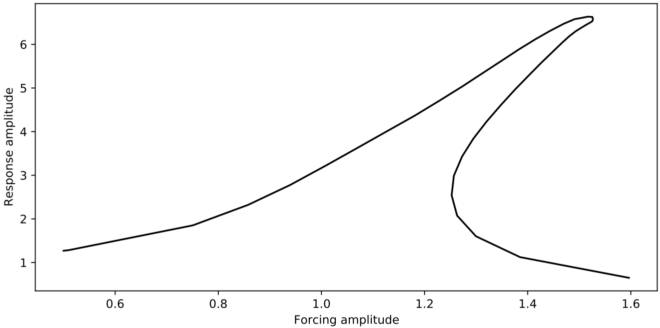
Continuation finishes early due to convergence failure

Fourier, SciPy solver, stepsize=0.2, $K_p = 2$



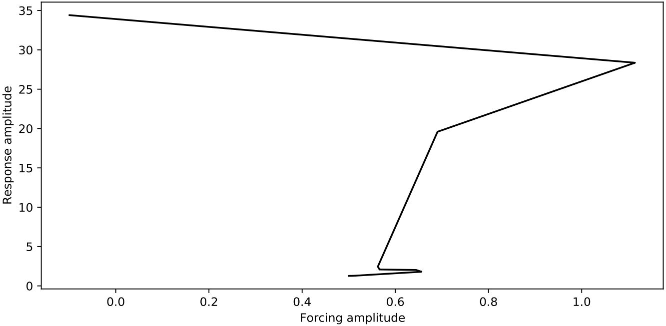
Success

Splines, SciPy solver, $K_p = 2.5$



Continuation terminates early due to lack of convergence

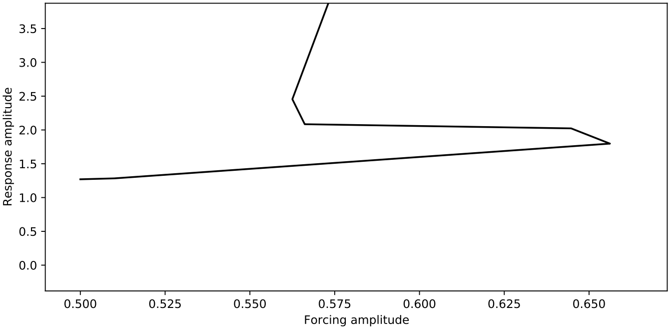
Splines, DIY Newton solver, $k_p = 2.5$



No idea what's going on here

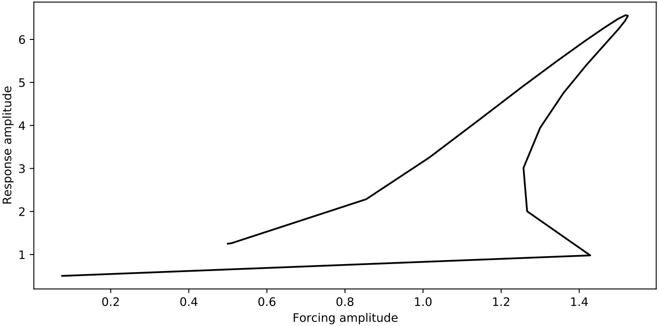
Splines, DIY Newton solver, $k_p = 2.5$

Zoomed in a bit



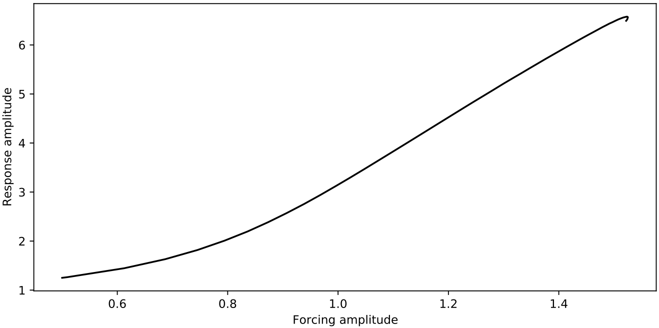
No idea what’s going on here

Fourier, SciPy solver, stepsize=1, $K_p = 2.5$



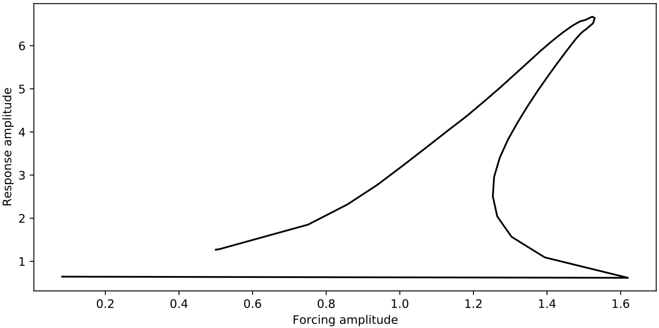
Continuation terminates early due to lack of convergence

Fourier, SciPy solver, stepsize=0.2, $K_p = 2.5$



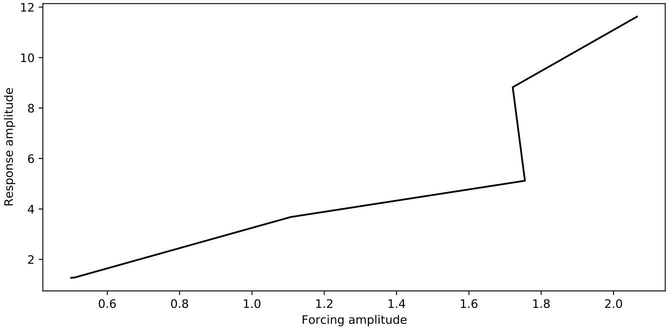
Failed due to lack of convergence

Splines, SciPy solver, $K_p = 2.75$



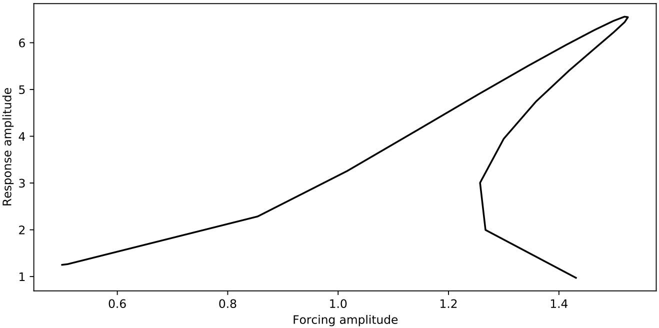
Fails to converge

Splines, DIY Newton solver, $k_p = 2.75$



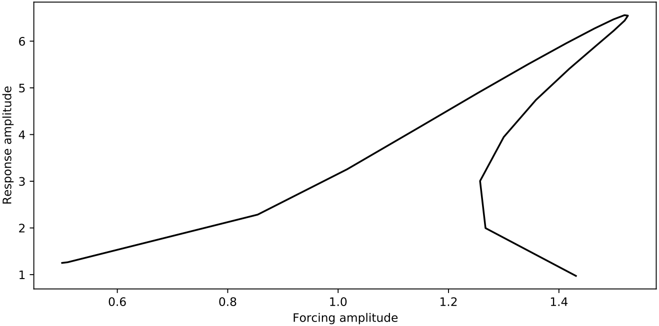
No idea what's going on here

Splines, SciPy solver, $K_p = 2.75$



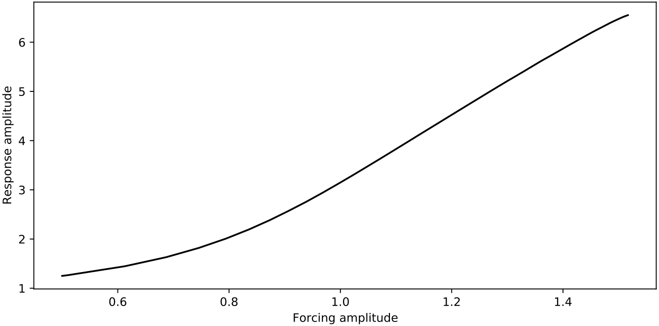
Fails to converge

Fourier, SciPy solver, stepsize=1, $K_p = 2.75$



Fails to converge

Fourier, SciPy solver, stepsize=0.2, $K_p = 2.75$



Fails to converge

Some notes on how collocation works

How collocation works

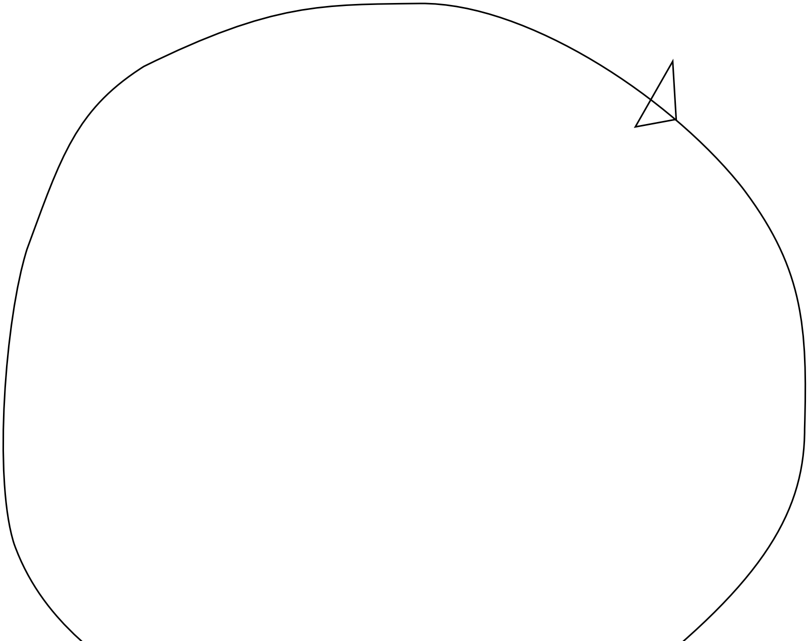
Using the standard collocation method

- Find LCs by solving a periodic BVP
  - Find a curve that's at some state at time 0, and back at that state at time T
  - State at 0, T forms a boundary value problem
- Rescale BVP to unit interval
- Split interval up into mesh
- Model solution as a polynomial within each mesh segment
- Find the polynomial coefficients that
  - exactly solve the BVP at the collocation points;
  - result in continuity between intervals;
  - result in periodicity;
  - also satisfy the phase constraint.
- Choose collocation points as (scaled) zeros of Legendre polynomials

Code was tested on Hopf normal form, as it only handles autonymous systems

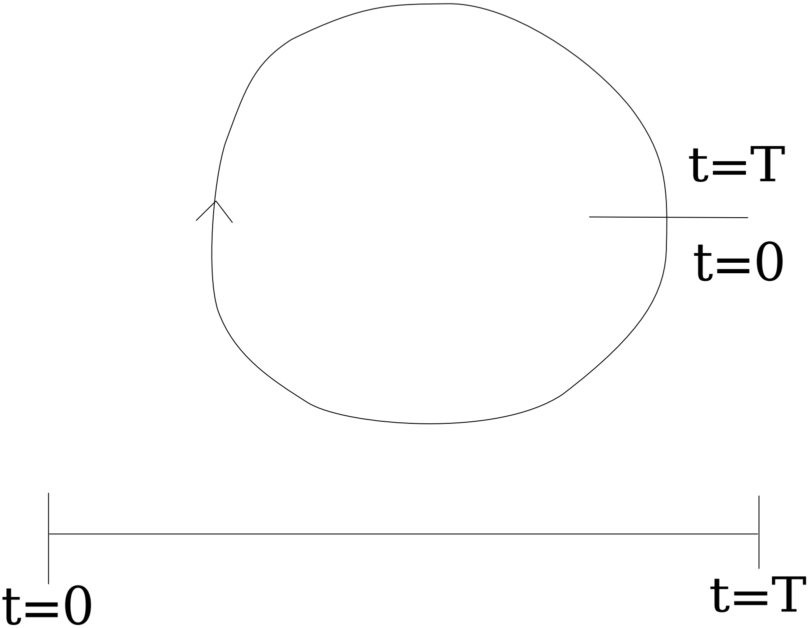
How collocation works

Consider a system containing a periodic orbit



How collocation works

Cut the orbit at some point; we then have a boundary-value problem



How collocation works

We now have BVP $\dot{x} = f(x), \quad x(0) = x(T)$

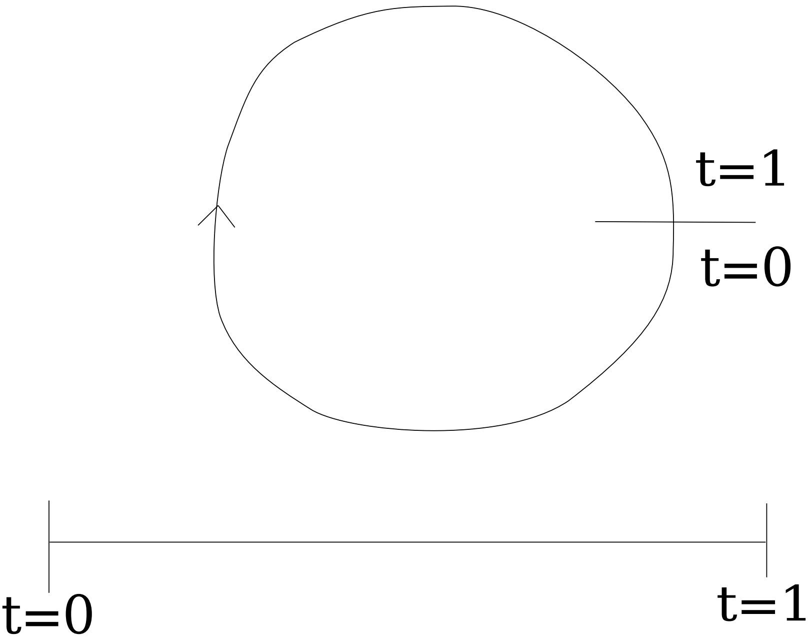
We could cut anywhere around the orbit, so we choose the cut so that the solution phase is as close as possible to that of some reference $v(t)$, achieved when

$$\int_0^T \langle x(t), \dot{v}(t) \rangle \mathrm{d}t = 0$$

Note that we don't have to explicitly solve for a cut-point; instead, we simply include this constraint within our problem, as a regularisation term.

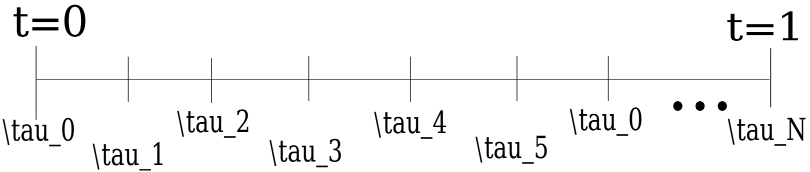
How collocation works

Let $\tau = Tt$. This rescales the BVP to the unit interval



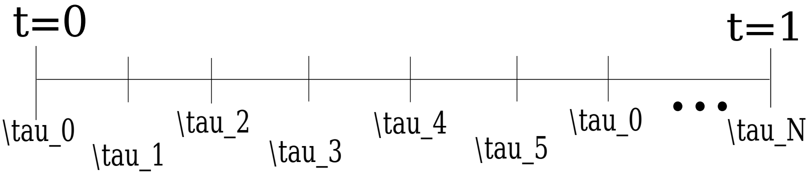
How collocation works

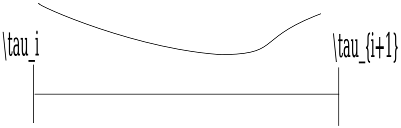
Split the BVP domain into a mesh



How collocation works

Use a set of Lagrange functions to define a polynomial solution approximation over each subinterval

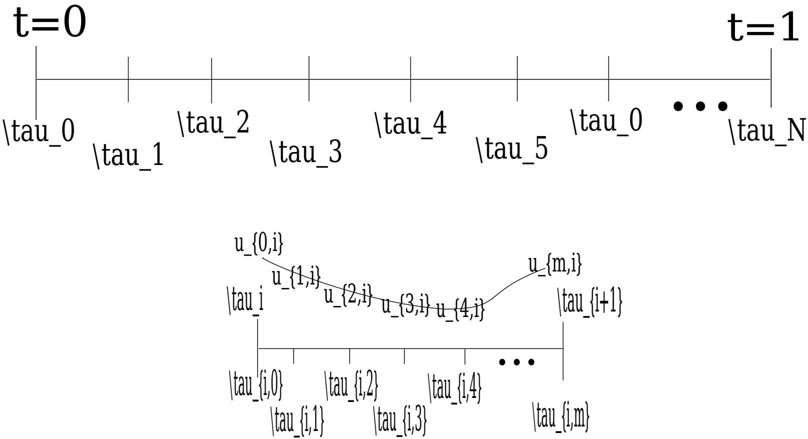




$$u^{(j)}(\tau) = \sum_{i=0}^{m} u^{j,i} l_{j,i}(\tau)$$

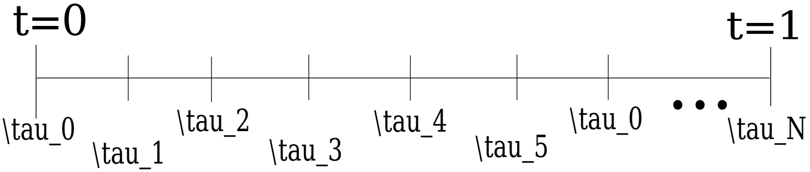
How collocation works

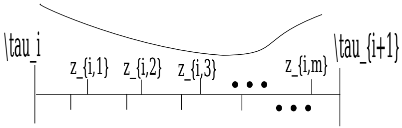
The coefficients $u_{j,i}$ are the value of the polynomial at a set of submesh-points $\tau_{i,j}$



How collocation works

We define a set of collocation points across this subinterval



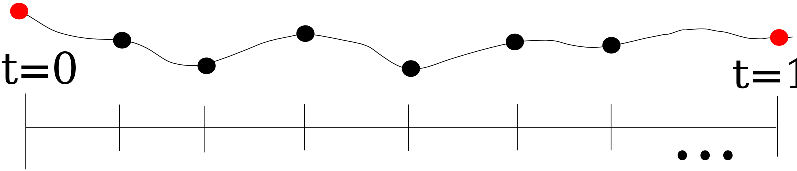


We require the BVP to be satisfied exactly at each collocation point

$$\sum_i u^{j,i} l_{j,i}(\tau) = f(\sum_i u^{j,i} l_{j,i}(\tau)), \quad \tau = z_{i,j}$$

How collocation works

This gives us mNn equations, for m collocation points, N collocation meshpoints, n dimensions.
We then require continuity between each interval's polynomials, and between the first and last (boundary) points



How collocation works

- This continuity requirement gives us an additional Nn equations.
- Finally, we have the periodicity constraint, giving a total of $mNn + nN + 1$ equations, for $(m+1)Nn$ unknowns $u^{j,i}$, and one unknown period T.
- Total of $mNn + nN + 1$ equations for $mNn + nN + 1$ unknowns, so we have a fully determined system!

BSpline Collocation

- Instead of placing a set of BSplines over every subinterval, we place a single BSpline curve over the entire domain.
- We choose the BSpline curve to be periodic, which removes the need for the Nn continuity / periodicity equations.
- The collocation points are distributed across this interval as usual.
- We instead have $n(N-1)$ unknowns, for the BSpline coefficients, and 1 unknown for the period.
- Choosing $N-1$ collocation points therefore gives us a full system of equations.
- In all cases, we choose the collocation points as the zeros of the Legendre polynomial of appropriate degree, rescaled to across the interval.