University of BRISTOL

[2020-10-26 Mon]

BSplines for CBC discretisation

Mark Blyth

Last time

Proposals for project ideas, with a focus on

- Efficiency
  - Avoiding high-dimensional discretisation
  - Producing fast prediction/correction steps
- Noise-robustness
  - Finding continuation solutions in the face of measurement noise
  - Whether or not to consider stochasticity

This time

Preliminary results for efficient splines-based discretisation

- Replace Fourier discretisation with splines discretisation
  - Goal: more noise-robust, lower dimensional

- Was facing issues with numerical simulations

- New results from Friday: it works!

Spline models

Currently testing spline discretisation; what is this?

- Standard continuation: set up a BVP for a PO
  - Continuation vector encodes discretised solution + regularisation constraints
  - Orthogonal collocation usually used
- Control-based continuation: set up a variational problem
  - Find a non-invasive control target
  - Use a solving algo on discretised signals
- Splines are maximally smooth piecewise-polynomial models
  - BSplines form a set of basis functions for spline models
  - BSpline coefficients are used as signal discretisation

The BSpline method

Key goal: noninvasive control

- For proportional control, zero tracking error means zero control action
- Noninvasive $\iff$ system output matches control target

Algo summary:

- Produce two initial discretisations by running the system uncontrolled

The BSpline method

Key goal: noninvasive control

- For proportional control, zero tracking error means zero control action
- Noninvasive $\iff$ system output matches control target

Algo summary:

- Produce two initial discretisations by running the system uncontrolled
- Use secant pseudo-arclength continuation to track solution under parameter change

The BSpline method

Key goal: noninvasive control

- For proportional control, zero tracking error means zero control action
- Noninvasive $\iff$ system output matches control target

Algo summary:

- Produce two initial discretisations by running the system uncontrolled
- Use secant pseudo-arclength continuation to track solution under parameter change
    1. Extract discretisation from continuation vector

The BSpline method

Key goal: noninvasive control

- For proportional control, zero tracking error means zero control action
- Noninvasive $\iff$ system output matches control target

Algo summary:

- Produce two initial discretisations by running the system uncontrolled
- Use secant pseudo-arclength continuation to track solution under parameter change
    1. Extract discretisation from continuation vector
    2. Translate discretisation into a control target

The BSpline method

Key goal: noninvasive control

- For proportional control, zero tracking error means zero control action
- Noninvasive $\iff$ system output matches control target

Algo summary:

- Produce two initial discretisations by running the system uncontrolled
- Use secant pseudo-arclength continuation to track solution under parameter change
  1. Extract discretisation from continuation vector
  2. Translate discretisation into a control target
  3. Run the system with that target

The BSpline method

Key goal: noninvasive control

- For proportional control, zero tracking error means zero control action
- Noninvasive $\iff$ system output matches control target

Algo summary:

- Produce two initial discretisations by running the system uncontrolled
- Use secant pseudo-arclength continuation to track solution under parameter change
  1. Extract discretisation from continuation vector
  2. Translate discretisation into a control target
  3. Run the system with that target
  4. Discretise the output

The BSpline method

Key goal: noninvasive control

- For proportional control, zero tracking error means zero control action
- Noninvasive $\iff$ system output matches control target

Algo summary:

- Produce two initial discretisations by running the system uncontrolled
- Use secant pseudo-arclength continuation to track solution under parameter change
    1. Extract discretisation from continuation vector
    2. Translate discretisation into a control target
    3. Run the system with that target
    4. Discretise the output
    5. Newton-solve for discretised input = discretised output

The BSpline method

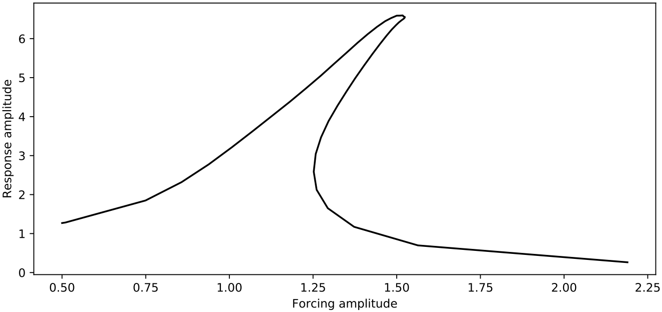
Key goal: noninvasive control

- For proportional control, zero tracking error means zero control action
- Noninvasive $\iff$ system output matches control target

Algo summary:

- Produce two initial discretisations by running the system uncontrolled
- Use secant pseudo-arclength continuation to track solution under parameter change
  1. Extract discretisation from continuation vector
  2. Translate discretisation into a control target
  3. Run the system with that target
  4. Discretise the output
  5. Newton-solve for discretised input = discretised output
  6. Repeat across target parameter range

Results



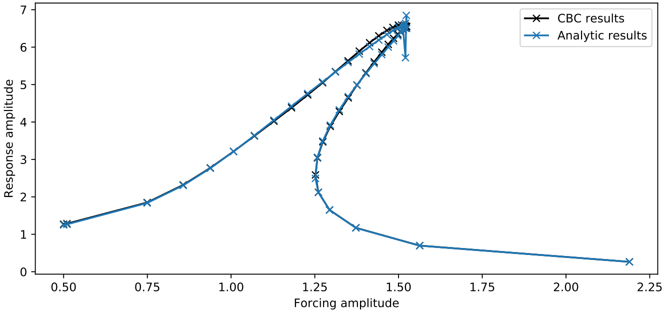
It works!

Results

- Results are for a harmonically forced Duffing oscillator
  - Validation: an solve analtically for frequency-response curves
  - Simplicity: system output is easy to Fourier- and spline-discretise

- Analytic results need computing through continuation
  - For simplicity I didn’t take the continuation route
  - That’s coming next!

Results



It works!

The hard parts

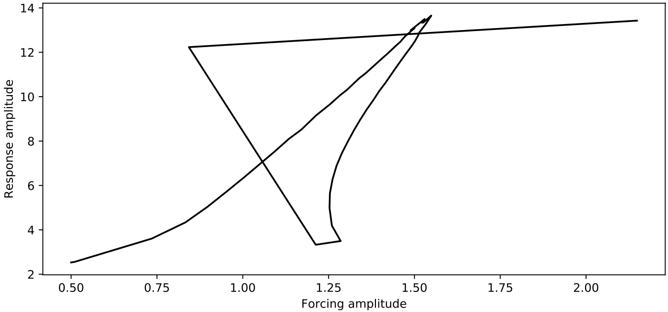
- All spline curves require exterior knots
  - 'Extra' control points placed outside the range of the data
  - Allows the spline curve to fit the data endpoints
- Periodic splines require careful treatment
  - Coefficient vectors have a special structure
  - Perturbations (prediction steps, finite differences) break that structure
  - Can make the coefficient vectors perturbation-robust quite easily, but it requires custom code
- Periodic splines take periodic exterior knots, and periodic coefficients for exterior BSplines
  - First k coeff's must equal last k coeffs
  - This is easy to handle; SciPy tries to be very general, and ends up handling it badly

The hard parts

My Newton solver doesn't solve the continuation equations very well

- Accepted solution vectors don't accurately solve the system
  - Convergence declared when solution stops changing
  - Converged vector gives a solution error of $\mathcal{O}(10^{-1})$
- My DIY solver 'jumps'
  - Solution vector normally takes small parameter-steps
  - Newton solver causes solution to take a very big parameter step, to somewhere wrong
- SciPy solvers overcome this...
  - ...however SciPy quasi-Newton solvers have the same issue!
  - Other methods work very well, but they're a black box
  - No idea what they're doing, or how or why

Jumping solutions



(Actually using slightly older code, but same results apply)

Solution existence and uniqueness

Under what conditions can we guarantee a solution to the CBC equations exists?

- Undiscretised case: solution definitely exists
  - Infinite Fourier discretisation is an exact representation of continuous case; solution must exist
  - Trucated Fourier is equivalent to infinite Fourier up to computational precision; solution *probably* exists

Solution to discretised equations must exist when discretisation is exact

- Can't guarantee splines are exact; how do we know if a solution exists?

Generally, when can we guarantee discretising won't cause the system to become unsolvable?

Next steps

1. Testing spline discretisation more
   - Try it out on a neuron model
   - Try to break it!

1. Understand the solver issues
   - Solvers are clearly crucial to good results
   - Need to understand where the Newton problems are coming from

1. Compare splines to other methods
   - Compare to Fourier, wavelets, collocation
   - Compare in terms of noise-robustness, efficiency, achievable accuracy, ease of use