Physics and Machine Learning Based Approaches to Stability Analysis and Control on DIII-D

by **R. Conlin**[1,2*]

with

J. Abbate[1,2]**, K. Erickson**[2]**, A. S. Glasser**[1,2]**, A. Wu**[1]**, A. Iqtidar**[1]**, E. Kolemen**[1,2]

[1] Princeton University
[2] Princeton Plasma Physics Laboratory

Presented remotely at
2020 APS-DPP Meeting
November 11th, 2020

Email:
* wconlin@pppl.gov

Mechanical and Aerospace Engineering
PRINCETON

PPPL

DIII-D NATIONAL FUSION FACILITY SAN DIEGO

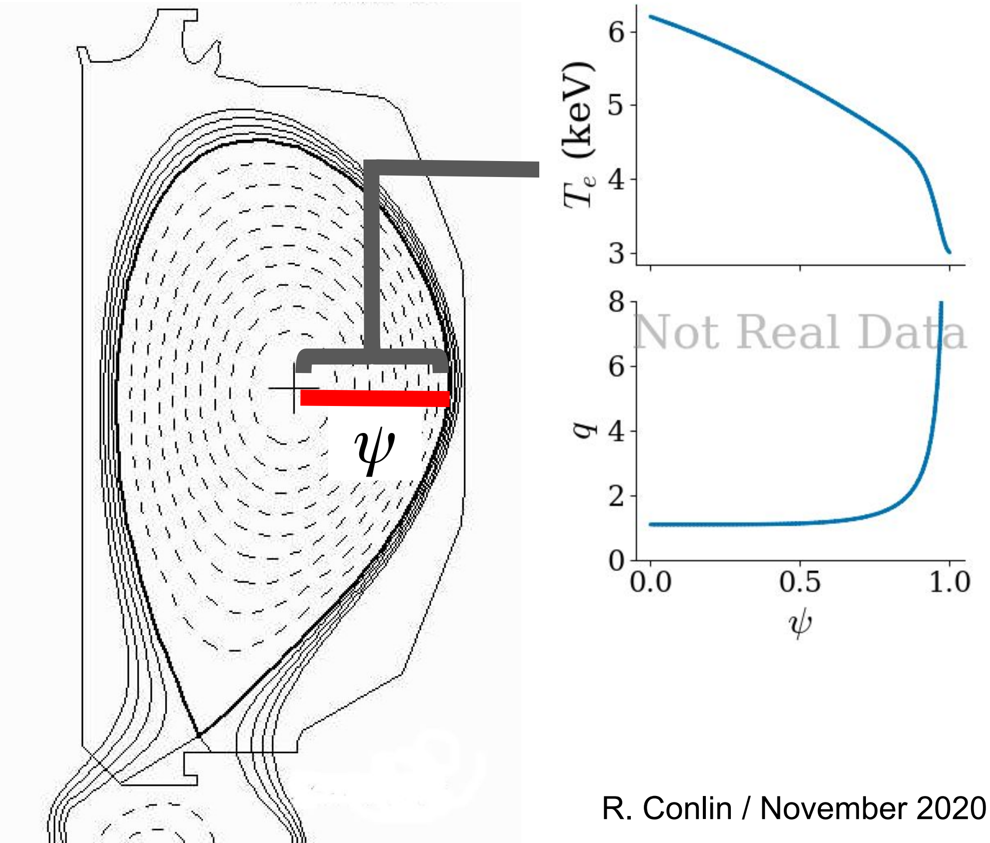
Outline

- Machine Learning to predict/control plasma state
  - What should control inputs be to achieve desired state?
- Using machine learning models in real time systems
  - How do we get a neural net onto plasma control system (PCS)?
- Physics based models to determine which states are best
  - Given a controller, which state should we aim for?

Outline

- Machine Learning to predict/control plasma state
  - What should control inputs be to achieve desired state?
- Using machine learning models in real time systems
  - How do we get a neural net onto plasma control system (PCS)?
- Physics based models to determine which states are best
  - Given a controller, which state should we aim for?

Transport Plasma State

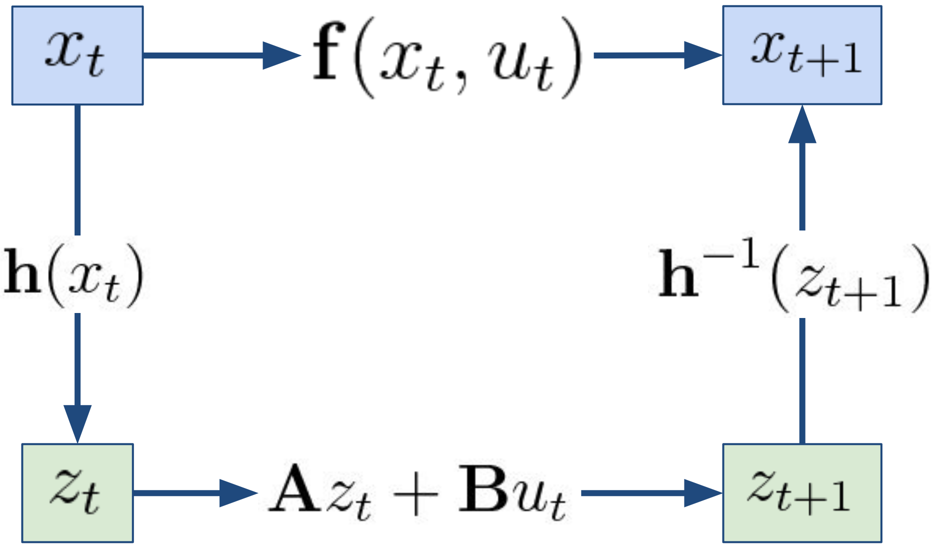


Full state of plasma determined by 1D profiles:

- Pressure (P)
- Current (J)
- Electron temperature and density (T_e, n_e)
- Ion temperature and density (T_i, n_i)
- Rotation (Ω)

Given state (and actuators), can we predict how plasma will evolve on transport timescales (~100-200ms)?

Transport is nonlinear - use ML to get linear model

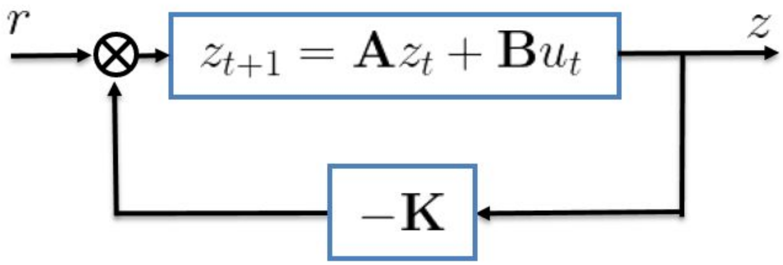


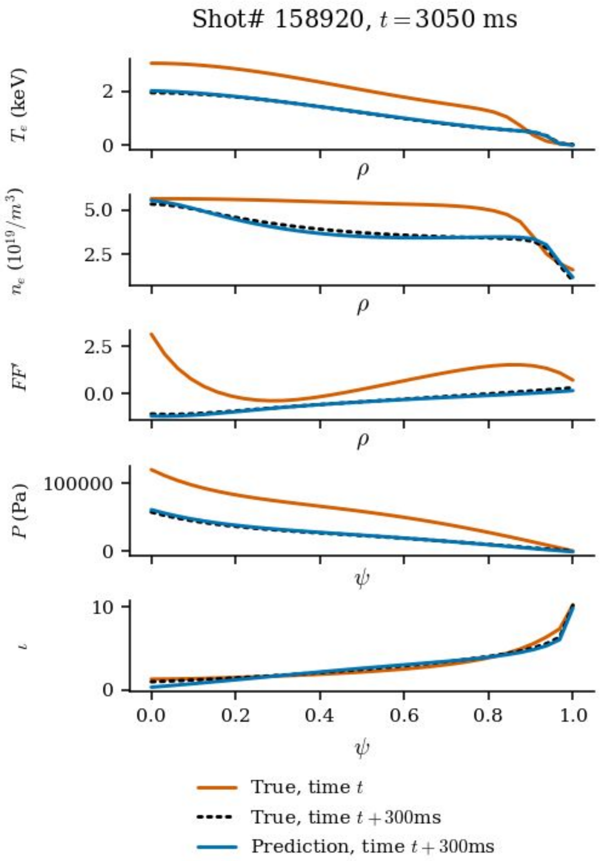
- Traditional ML*: Learn **f,** but model predictive control with nonlinear model is expensive, inefficient.
- Solution: use **Linearly Recurrent Autoencoder Network** (LRAN) to learn linear embedding of nonlinear dynamics
- Functions **h** and **h**$^{-1}$ parameterized by neural networks
- Learned along with matrices **A**, **B**
- Gives linear model for dynamics, so we can use robust methods for linear optimal control

* see:
- Abbate, ZO04.00006 Data-Driven Profile Prediction,
- Jalalvand, GP19.00024 Hyper-dimensional time-series data analysis with reservoir computing networks to predict plasma profiles in tokamak

LRAN: high accuracy, easy robust control design

- Model trained on experimental data from DIII-D 2013-2018
- After model tuning, can get similar performance to more advanced models
- Currently developing finite horizon linear optimal controller for tests on DIII-D



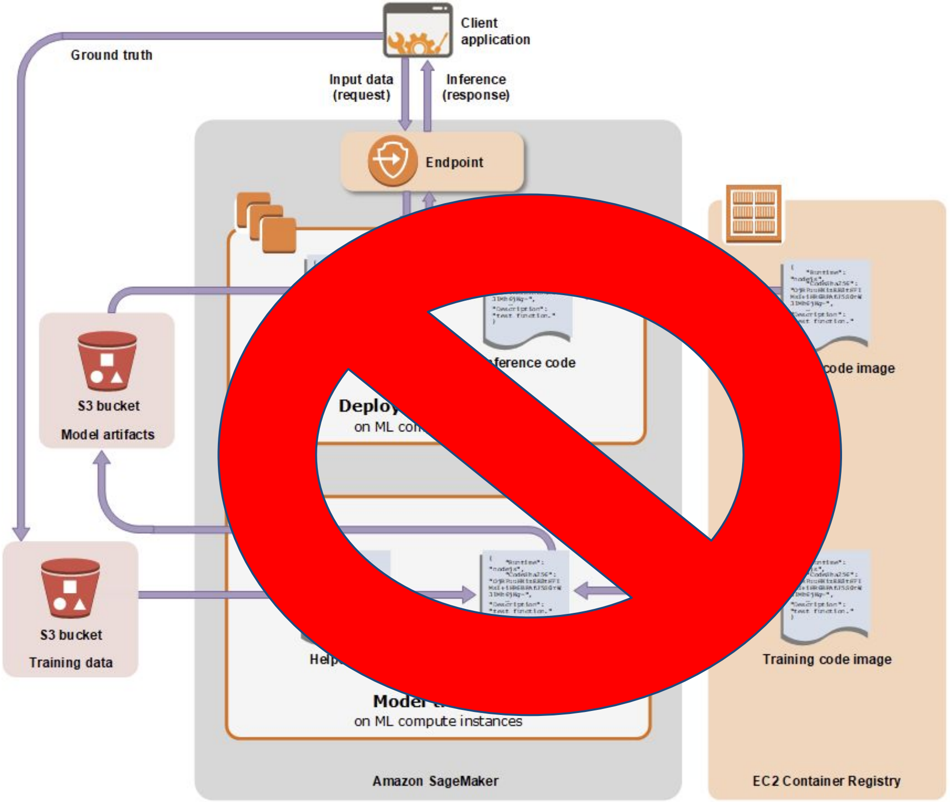


Outline

- Machine Learning to predict/control plasma state
    - What should control inputs be to achieve desired state?
- Using machine learning models in real time systems
    - How do we get a neural net onto plasma control system (PCS)?
- Physics based models to determine which states are best
    - Given a controller, which state should we aim for?

How to deploy machine learning models for control?

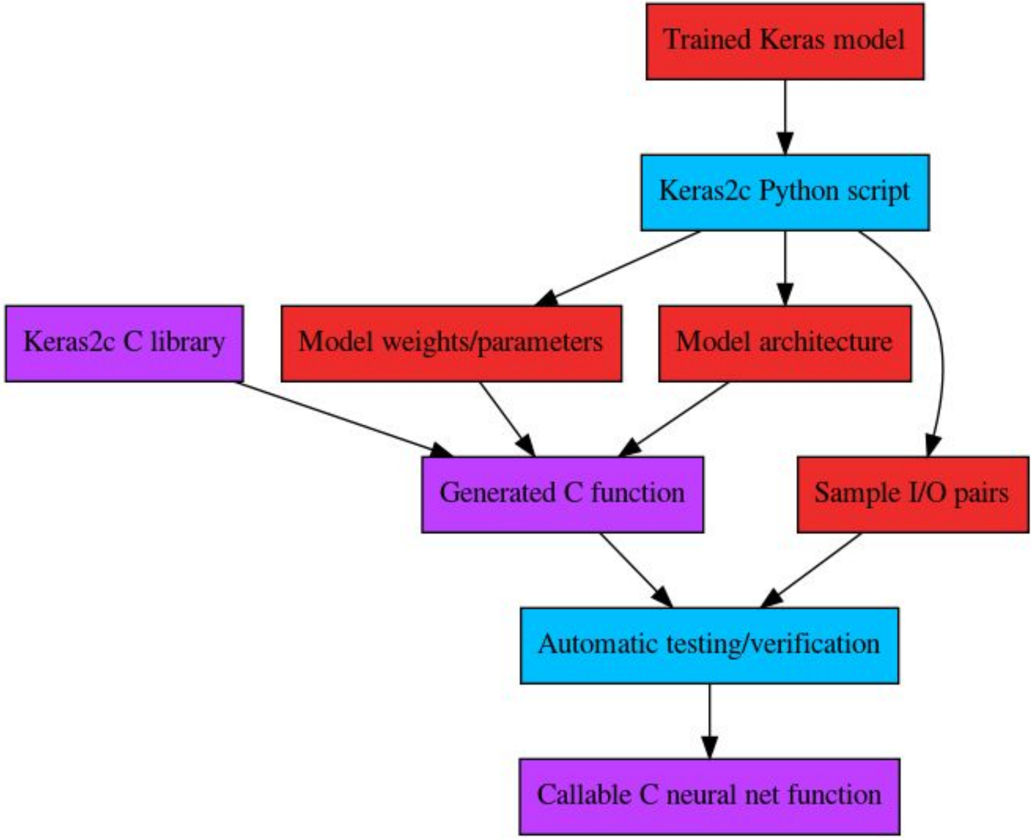
- Current method for deploying ML models based around mobile + web applications
- Generally involve communicating with process running on remote server
  - Large latency
  - Non-deterministic behavior
  - **Not safe for real-time applications**
- Other option: recode entire model by hand
  - Time consuming
  - Error prone



Keras2c: full automated conversion / code generation

Script/Library for converting Keras neural nets to C functions

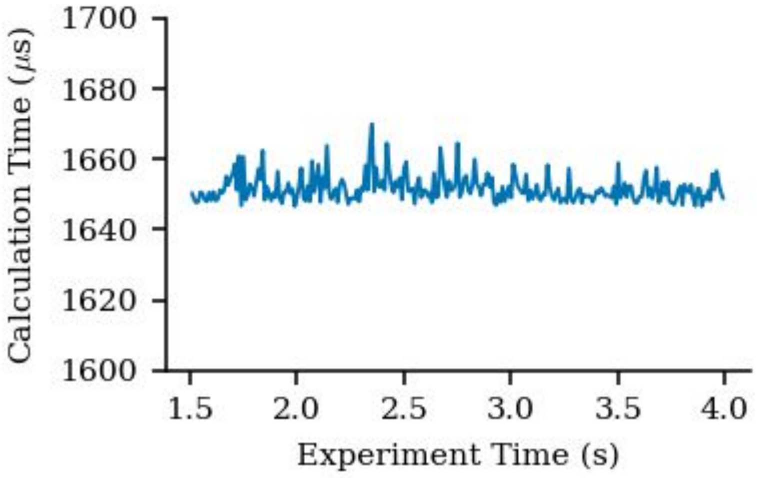
- Designed for simplicity and real time applications
- Core functionality only ~1500 lines
- Generates self-contained C function, no external dependencies
- Supports full range of operations & architectures
- Fully automated conversion & testing



Real-time applications: DIII-D Plasma Control

- Example timing shown for neural net predicting plasma transport
  - 30 convolutional layers of varying size
  - 2 recurrent LSTM layers
  - Dozens of reshaping/padding/merging operations
  - Multi-input/multi-output model with branching internal structure
  - Total 45,485 parameters
- Mean time 1.65 ms*
- Worst case jitter 23 μs, rms 3.75 μs

*Also includes time to gather input data from other processes and pre-processing



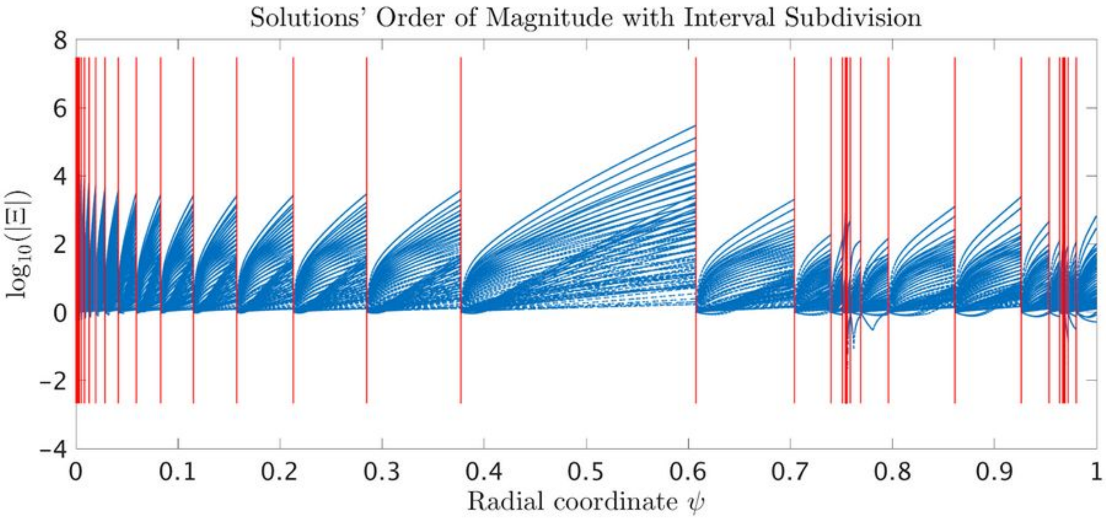
Outline

- Machine Learning to predict/control plasma state
  - What should control inputs be to achieve desired state?
- Using machine learning models in real time systems
  - How do we get a neural net onto plasma control system (PCS)?
- Physics based models to determine which states are best
  - Given a controller, which state should we aim for?

STRIDE: Real Time δW Calculations

$$\delta W = \frac{1}{2}\int_\Omega \mathrm{d}\mathbf{x}\left[Q^2 + \mathbf{J}\cdot\boldsymbol{\xi}\times\mathbf{Q} + (\boldsymbol{\xi}\cdot\boldsymbol{\nabla}P)(\boldsymbol{\nabla}\cdot\boldsymbol{\xi}) + \gamma P(\boldsymbol{\nabla}\cdot\boldsymbol{\xi})^2\right]$$

- δW < 0 → MHD instability
- Quadratic Lagrangian gives Linear Euler-Lagrange equation
- Linear E-L can be domain decomposed using state transition matrices



$$\mathbf{x}'(\psi) = \mathbf{L}(\psi)\mathbf{x}(\psi)$$

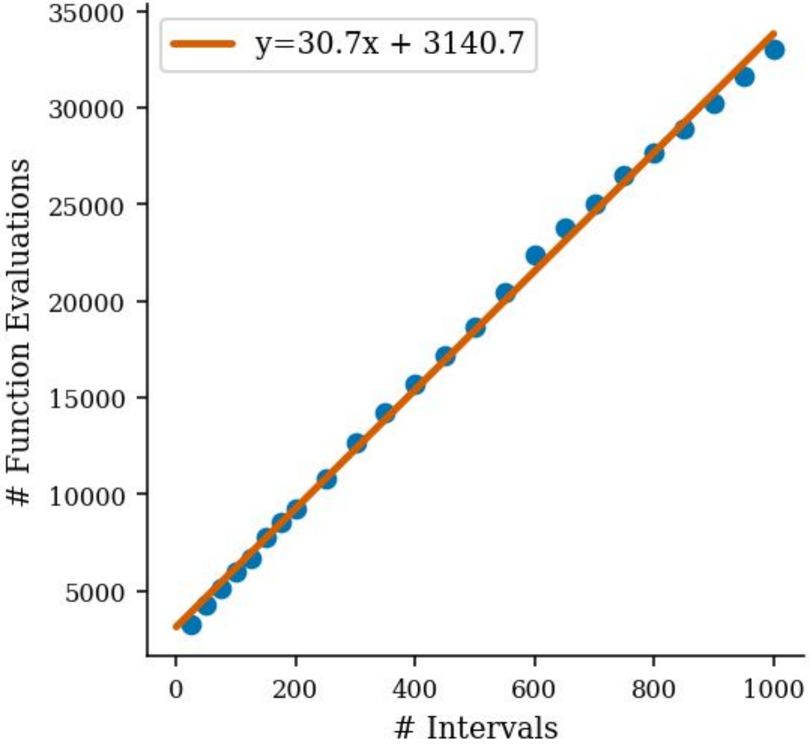
$$\boldsymbol{\Phi}'(\psi) = \mathbf{L}(\psi)\boldsymbol{\Phi}(\psi)$$

$$\mathbf{x}(\psi_2) = \boldsymbol{\Phi}(\psi_2,\psi_0)\mathbf{x}(\psi_0) = \boldsymbol{\Phi}(\psi_2,\psi_1)\boldsymbol{\Phi}(\psi_1,\psi_0)\mathbf{x}(\psi_0)$$

Easy parallelization → fast (real time) stability calculations

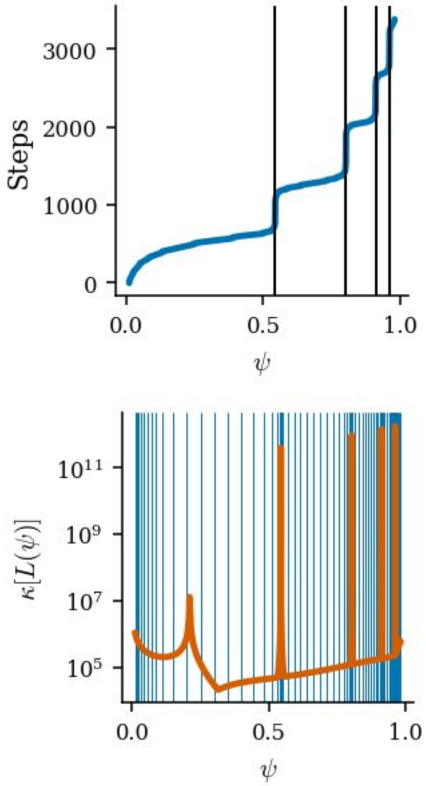
Adaptive multistep integration scales poorly with many threads

- Previous approach used ZVODE (adaptive multistep method) to integrate on each interval
- Adaptive step size takes extra unnecessary steps in stiff regions
- Multistep method not self starting, needs extra function evaluations on each interval.
- **Adding more intervals to balance threads adds 1000s of function evaluations**
- Compute time ~300 ms at best on 72 core CPU



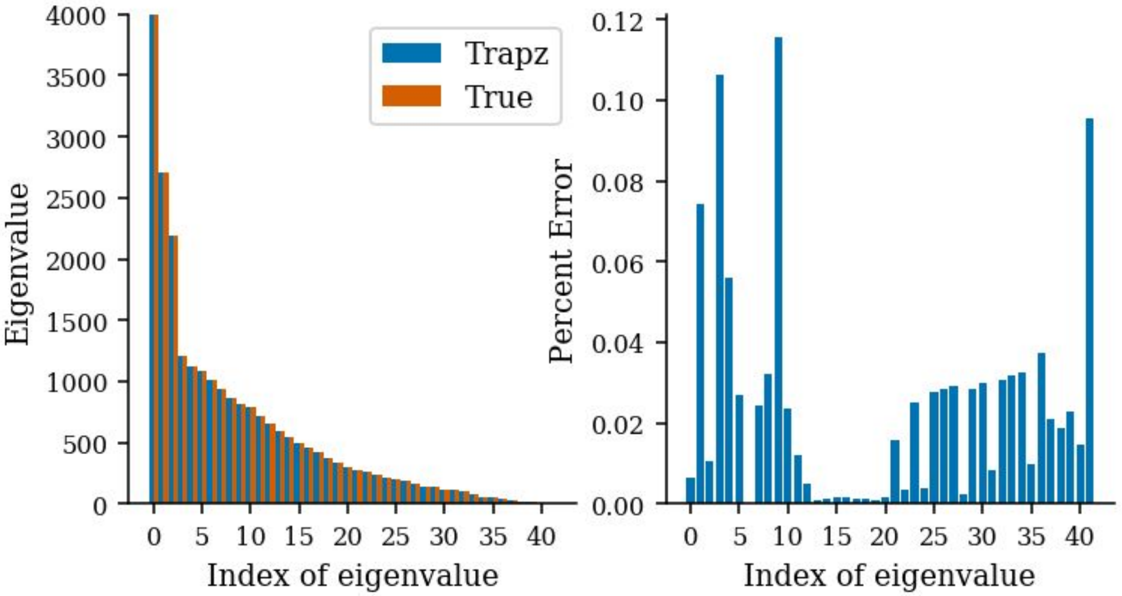
Extreme parallelization - fixed steps, tuned intervals

- Use 1 trapezoidal step per interval with optimized interval division
- Binary reduction to combine solutions in ~$\text{Log}_2(\text{N})$ time
- Know we need to take smaller steps closer to rational surfaces
  - Assume step size $h \sim 1/\kappa$ where κ is some measure of stiffness
  - Fit a function of the form $\kappa = \sum_s \frac{\alpha}{1+\beta|\psi - \psi_s|}$
    - s = index of singularity,
    - ψ_s = location of singularities
    - α, β = coefficients to optimize



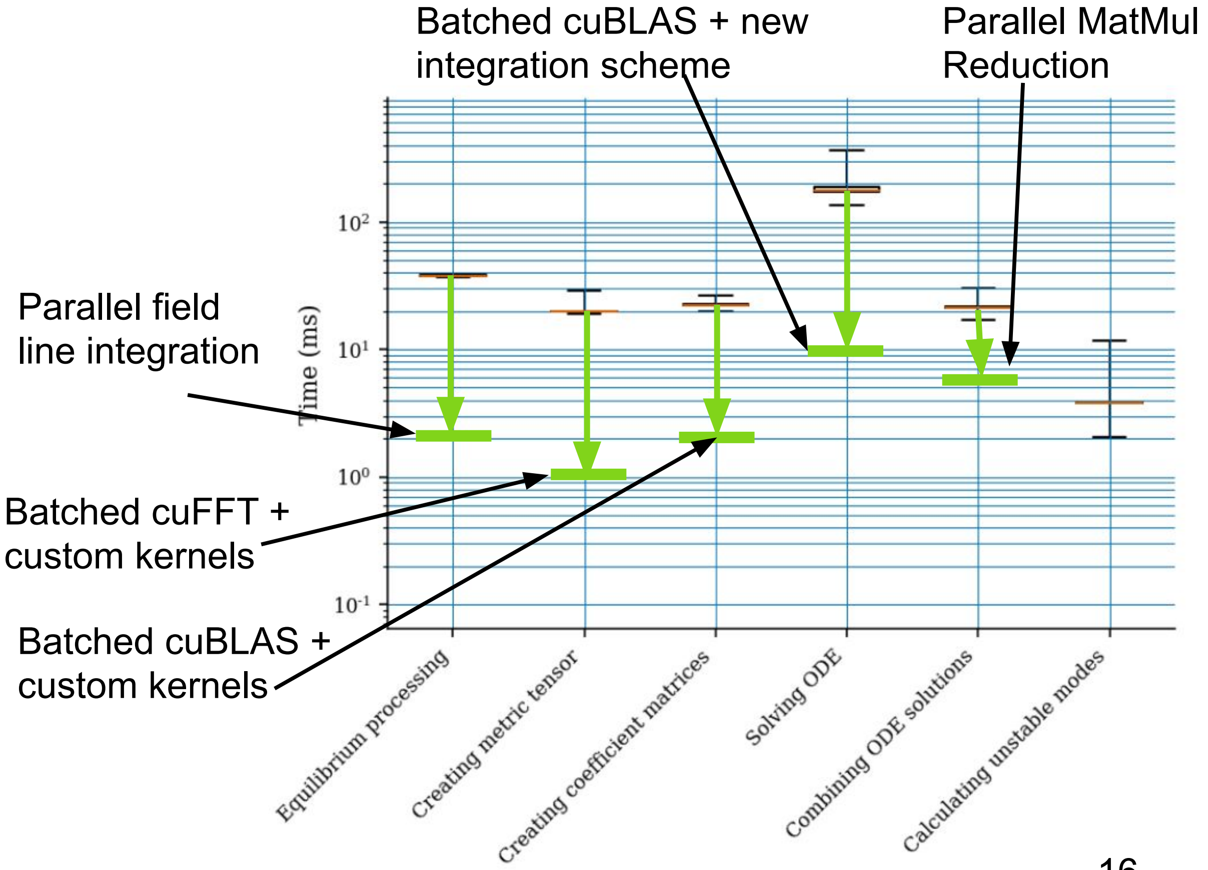
Significant speedup, minimal error

- Trapezoidal method reduces integration cost by ~10x, with only 0.1% error in eigenvalues of plasma response matrix
- Implemented in PCS, achieves **calculation times < 100 ms**
- Ideal for real time analysis
    - Integrating with Proximity Control to steer away from stability boundary
    - But need faster still for model based predictive control



STRIDE GPU for predictive stability analysis

- GPU implementation under development
- Projected to achieve <**20ms calculation time**
- Combined with state prediction, hope to **predict instabilities 100s of ms before they occur**



Summary

- **Autoencoders can learn linear embedding for robust control design**
  - S. Otto, C. Rowley: “Linearly recurrent autoencoder networks for learning dynamics”, *SIAM Journal on Applied Dynamical Systems* (2019)
  - J. Abbate, R. Conlin, E. Kolemen: “Data-Driven Profile Prediction for DIII-D”, *Nuclear Fusion* (under review)
  - A. Jalalvand, J. Abbate, R. Conlin, G. Verdoolaege, E. Kolemen (2020), "Real-Time and Adaptive Reservoir Computing with an Application to Profile Prediction in Fusion Plasma", *IEEE Transactions on Neural Networks and Learning Systems*. (Under Review)
- **Keras2c allows automatic conversion of neural networks to real time C code**
  - https://github.com/f0uriest/keras2c
  - R. Conlin, K. Erickson, J. Abbate, E. Kolemen: "Keras2c: A library for converting Keras neural networks to real-time compatible C", *Engineering Applications of Artificial Intelligence* (under review)
- **STRIDE calculates ideal MHD stability in real time**
  - A.S. Glasser, E. Kolemen, A.H. Glasser: “A Riccati solution for the ideal MHD plasma response with applications to real-time stability control”, *Physics of Plasmas* (2018)
  - A.S. Glasser, E. Kolemen: “A robust solution for the resistive MHD toroidal Δ' matrix in near real-time”, *Physics of Plasmas* (2018)
  - A.S. Glasser, A.H. Glasser, R. Conlin, E. Kolemen: “An ideal MHD δW stability analysis that bypasses the Newcomb equation”, *Physics of Plasmas* (2020)

Koopman operator theory

Consider a nonlinear discrete time system:

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t) \tag{1}$$

with state $\boldsymbol{x} \in \mathbb{R}^n$ and continuous map $\boldsymbol{f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$

Let $\boldsymbol{g}(\boldsymbol{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be an observable of the system. The collection of all observables form a linear vector space $\mathcal{G}$.

Define the Koopman operator U as a linear transformation on this vector space as follows:

$$U\boldsymbol{g}(\boldsymbol{x}_t) = \boldsymbol{g} \circ \boldsymbol{f}(\boldsymbol{x}_t) = \boldsymbol{g}(\boldsymbol{x}_{t+1}) \tag{2}$$

Where $\circ$ denotes the composition operator $(z \circ y(x) = z(y(x)))$. The linearity follows directly from the linearity of the composition operator:

$$U[\boldsymbol{g}_1 + \boldsymbol{g}_2](\boldsymbol{x}) = [\boldsymbol{g}_1 + \boldsymbol{g}_2] \circ \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{g}_1 \circ \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}_2 \circ \boldsymbol{f}(\boldsymbol{x}) = U\boldsymbol{g}_1(\boldsymbol{x}) + U\boldsymbol{g}_2(\boldsymbol{x}) \tag{3}$$

Koopman operator theory

Thus, we have transformed our original nonlinear system $\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t)$ into a linear system in the observables of $\boldsymbol{x}$, given by $\boldsymbol{g}(\boldsymbol{x}_{t+1}) = U\boldsymbol{g}(\boldsymbol{x}_t)$. However, this new linear system is infinite dimensional, due to the infinite dimensionality of the vector space $\mathcal{G}$.

However, because the Koopman operator is linear, we can seek to find its eigenvalues λ_j and eigenfunctions ϕ_j, which satisfy

$$U^t \phi_j(\boldsymbol{x}) = \lambda_j^t \phi_j(\boldsymbol{x}) \tag{4}$$

And assuming that the eigenfunctions span $\mathcal{G}$, we can decompose any observable as

$$\boldsymbol{g}(\boldsymbol{x}) = \sum_k \boldsymbol{g}_k \phi_k(\boldsymbol{x}) \tag{5}$$

We can define an observable to be the full state $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{x}$, whose Koopman decomposition is given by

$$\boldsymbol{x} = \sum_j \boldsymbol{\xi}_k \phi_k(\boldsymbol{x}) \tag{6}$$

LRAN theory

The evolution of the state is then given by

$$\boldsymbol{x}_t = \sum_j \boldsymbol{\xi_j} \lambda_j^t \phi_j(\boldsymbol{x}_0) \tag{7}$$

We can then interpret the autoencoder $\boldsymbol{h}$ as learning the Koopman eigenfunctions ϕ_j, and the learned matrix $\boldsymbol{A}$ as a low dimensional approximation to the Koopman operator, with eigenvalues λ_j and eigenvectors $\boldsymbol{\xi}_j$

We train the autoencoder to both minimize the traditional residual in $\boldsymbol{x}$, as well as the recurrent residual in the latent space $\boldsymbol{z} = \boldsymbol{h}(\boldsymbol{x})$

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_t \left(\boldsymbol{x}_t - \boldsymbol{h}^{-1}(\boldsymbol{h}(\boldsymbol{x}_t, \boldsymbol{\theta}), \boldsymbol{\theta})\right)^2 + \left(\boldsymbol{z}_{t+1} - (\boldsymbol{A}(\boldsymbol{\theta})\boldsymbol{z}_t + \boldsymbol{B}(\boldsymbol{\theta})\boldsymbol{u}_t)\right)^2 \tag{8}$$

I am visiting another poster session and will return at 4:15 EST