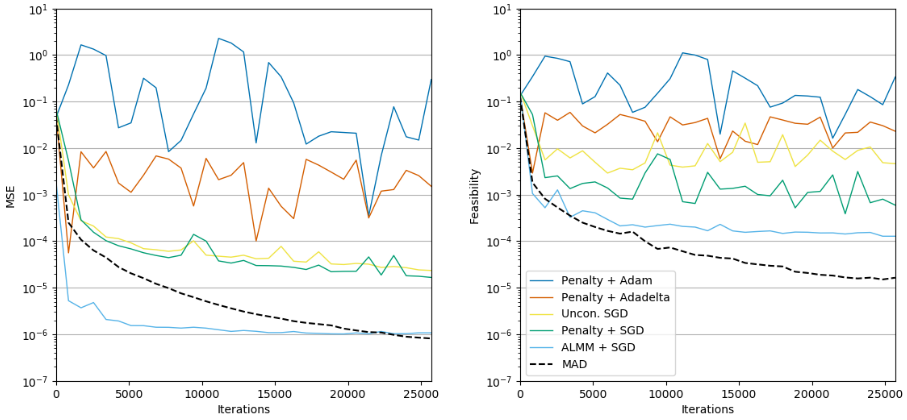
TOWARD CONSTRAINED OPTIMIZATION IN MACHINE LEARNING:

An Error-Tolerant Multisecant Method for Training PINNs



Alp Dener (Presenter)
Argonne National Laboratory

Todd Munson
Argonne National Laboratory

Data and DNN:
M. Andres Miller
R. Michael Churchill
Choong-Seock Chang

SIAM CSE 2021

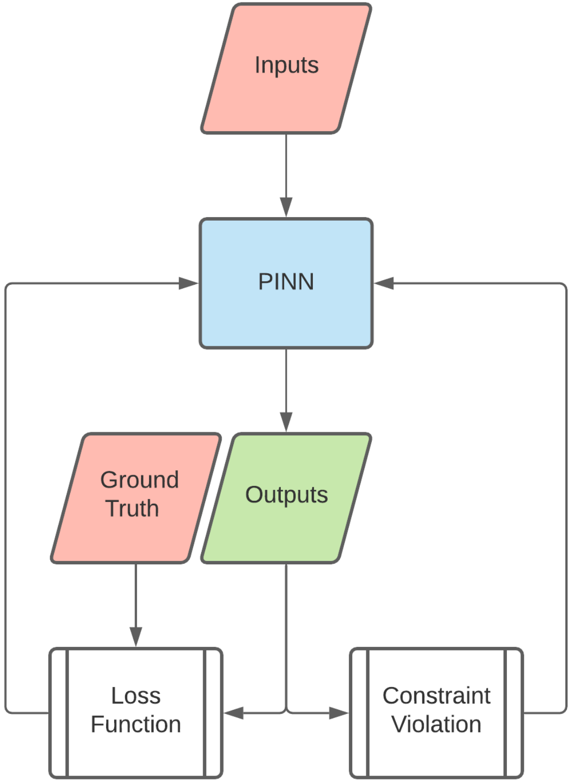
March 1, 2021

WHAT ARE PINNs?

- "Neural networks trained to solve supervised learning tasks while respecting any given laws of physics…" (Raissi et. al., 2019)

- NNs for approximating physical processes are not new, but most efforts treat NNs as "black box" function estimators

- PINNs seek to incorporate information about the underlying physics into the NN architecture or the training problem



TWO CLASSES OF PINNs…

”Hard” Constrained

NN architecture encodes constraints information

- Untrainable projection layer(s)
  (Mohan et. al., 2020)
- Embedded governing eqns.
  (Raissi et. al., 2019)

Requires knowledge of underlying physics

”Soft” Constrained

Training problem reformulated as constrained optimization

- Penalized loss functions
  (Erichson et. al., 2019)
  (Wu et. al., 2019)
  (Raissi et. al., 2019)

Difficulty tuning penalty term
Feasibility is not guaranteed

TWO CLASSES OF PINNs...

"Hard" Constrained

NN architecture encodes constraints information

- Untrainable projection layer(s) (Mohan et. al., 2020)
- Embedded governing eqns. (Raissi et. al., 2019)

Requires knowledge of underlying physics

"Soft" Constrained

Training problem reformulated as constrained optimization

- Penalized loss functions
  (Erichson et. al., 2019)
  (Wu et. al., 2019)
  (Raissi et. al., 2019)

Difficulty tuning penalty term
Feasibility is not guaranteed

WHY "SOFT" CONSTRAINTS?

Advantages:

- No assumptions about governing equations, quantities of interest, or constraint properties

- Easy to plug-and-play into different applications

- Domain experts do not need to understand ML and vice versa

Challenges:

- State-of-the-art is penalty methods sometimes fail to train! (Wang et. al. 2020)
  - Tuning penalty factors
  - Scaling discrepancies between loss and constraints

WHY "SOFT" CONSTRAINTS?

Advantages:

- No assumptions about governing equations, quantities of interest, or constraint properties
- Easy to plug-and-play into different applications
- Domain experts do not need to understand ML and vice versa

Challenges:

- State-of-the-art is penalty methods sometimes fail to train! (Wang et. al. 2020)
    - Tuning penalty factors
    - Scaling discrepancies between loss and constraints

We need better training methods!

OUTLINE

Review

- Training with Constraints
- Dealing with Large Data Sets
- Penalty Methods

Constrained Training

- Sequential Quadratic Programming
- Quasi-Newton Approximations
- Multisecant Method

Test Cases

- MNIST Image Classification Problem (Unconstrained)
- Approximating the Fokker-Planck-Landau Collision Operator

TRAINING WITH CONSTRAINTS

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \mathcal{J}(x;y) = \|\mathcal{M}(p;y) - \mathcal{R}(y)\|_2^2 \\ \text{subject to} \quad & \mathcal{C}(\mathcal{M}(x;y)) = 0 \end{aligned}$$

- $\mathcal{J}(x; y)$ – mean squared error (MSE) loss function
- $\mathcal{M}(x; y)$ – neural network model
- $\mathcal{C}(\mathcal{M}(x; y))$ – constraints on model output
- x – network weights/parameters
- $\mathcal{R}(y)$ – "ground truth" function/process to be approximated by the network

TRAINING WITH CONSTRAINTS

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \mathcal{J}(x;y) = \|\mathcal{M}(p;y) - \mathcal{R}(y)\|_2^2 \\ \text{subject to} \quad & \mathcal{C}(\mathcal{M}(x;y)) = 0 \end{aligned}$$

`scipy.optimize` can solve a full-batch problem

- $\mathcal{J}(x;y)$ – mean squared error (MSE) loss function
- $\mathcal{M}(x;y)$ – neural network model
- $\mathcal{C}(\mathcal{M}(x;y))$ – constraints on model output
- x – network weights/parameters
- $\mathcal{R}(y)$ – “ground truth” function/process to be approximated by the network

DEALING WITH LARGE DATA SETS

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \mathcal{J}(x;y) = \|\mathcal{M}(p;y) - \mathcal{R}(y)\|_2^2 \\ \text{subject to} \quad & \mathcal{C}(\mathcal{M}(x;y)) = 0 \end{aligned}$$

Split the data into randomized batches of size N_b

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \hat{\mathcal{J}}(x;y) = \frac{1}{N_b} \sum_{i}^{N_b} \|\mathcal{M}(x;y_i) - \mathcal{R}(y_i)\|_2^2 \\ \text{subject to} \quad & \hat{\mathcal{C}}(\mathcal{M}(x;y)) = \frac{1}{N_b} \sum_{i}^{N_b} \mathcal{C}(\mathcal{M}(x;y_i)) = 0 \end{aligned}$$

DEALING WITH LARGE DATA SETS

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \mathcal{J}(x;y) = \|\mathcal{M}(p;y) - \mathcal{R}(y)\|_2^2 \\ \text{subject to} \quad & \mathcal{C}(\mathcal{M}(x;y)) = 0 \end{aligned}$$

Split the data into randomized batches of size N_b

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \hat{\mathcal{J}}(x;y) = \frac{1}{N_b}\sum_i^{N_b} \|\mathcal{M}(x;y_i) - \mathcal{R}(y_i)\|_2^2 \\ \text{subject to} \quad & \hat{\mathcal{C}}(\mathcal{M}(x;y)) = \frac{1}{N_b}\sum_i^{N_b} \mathcal{C}(\mathcal{M}(x;y_i)) = 0 \end{aligned}$$

Conventional constrained optimization methods cannot solve this!

PENALTY METHODS

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \hat{\mathcal{J}}(x;y) \\ \text{subject to} \quad & \hat{\mathcal{C}}(\mathcal{M}(x;y)) = 0 \end{aligned}$$

$$\Downarrow$$

$$\underset{x}{\text{minimize}} \quad \hat{\mathcal{J}}(x;y) + \frac{\mu}{2}\|\hat{\mathcal{C}}(\mathcal{M}(x;y))\|_2^2$$

- Add a scalar measure of constraint violation into the loss function
- Converts into unconstrained problem
- ℓ_2-penalty example shown
- Parameter μ determines emphasis on constraint

SEQUENTIAL QUADRATIC PROGRAMMING

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \hat{\mathcal{J}}(x;y) \\ \text{subject to} \quad & \hat{\mathcal{C}}(\mathcal{M}(x;y)) = 0 \end{aligned}$$

Formulate the Lagrangian: $\mathcal{L} = \hat{\mathcal{J}} + \lambda^T \hat{\mathcal{C}}$

Differentiate for first-order optimality conditions:

$$\begin{aligned} \nabla_x \mathcal{L} = \nabla_x \hat{\mathcal{J}} + \lambda^T \nabla_x \hat{\mathcal{C}} &= 0 \\ \nabla_\lambda \mathcal{L} = \hat{\mathcal{C}} &= 0 \end{aligned}$$

Apply Newton's method:

$$\begin{bmatrix} H & A \\ A^T & 0 \end{bmatrix} \begin{pmatrix} \Delta p \\ \Delta \lambda \end{pmatrix} = \begin{pmatrix} -\nabla_x \mathcal{L} \\ -\hat{\mathcal{C}} \end{pmatrix} \qquad \begin{aligned} H &= \nabla^2_{xx} \mathcal{L} \\ A &= \nabla_x \hat{\mathcal{C}} \end{aligned}$$

SEQUENTIAL QUADRATIC PROGRAMMING

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \hat{\mathcal{J}}(x;y) \\ \text{subject to} \quad & \hat{\mathcal{C}}(\mathcal{M}(x;y)) = 0 \end{aligned}$$

Formulate the Lagrangian: $\mathcal{L} = \hat{\mathcal{J}} + \lambda^T \hat{\mathcal{C}}$

Differentiate for first-order optimality conditions:

$$\begin{aligned} \nabla_x \mathcal{L} = \nabla_x \hat{\mathcal{J}} + \lambda^T \nabla_x \hat{\mathcal{C}} &= 0 \\ \nabla_\lambda \mathcal{L} = \hat{\mathcal{C}} &= 0 \end{aligned}$$

Apply Newton's method:

Avoid computing! $$\begin{bmatrix} H & A \\ A^T & 0 \end{bmatrix} \begin{pmatrix} \Delta p \\ \Delta \lambda \end{pmatrix} = \begin{pmatrix} -\nabla_x \mathcal{L} \\ -\hat{\mathcal{C}} \end{pmatrix} \qquad \begin{aligned} H &= \nabla^2_{xx} \mathcal{L} \\ A &= \nabla_x \hat{\mathcal{C}} \end{aligned}$$

QUASI-NEWTON APPROXIMATIONS

$$W_{k+1} = \begin{cases} \underset{W}{\text{minimize}} & \|W - W_k\| \\ \text{subject to} & W(g_k - g_{k-1}) = x_k - x_{k-1} \end{cases}$$

$$W = \left(\nabla^2_{xx} f(x)\right)^{-1}$$
$$g = \nabla_x f(x)$$

- Quasi-Newton methods iteratively approximate the inverse-Hessian but they do not address constraints
- Cannot compute full-batch constraint Jacobians for large data sets so we need to approximate

Can we approximate the entire matrix (Hessian + constraint Jacobians) using inaccurate first-order information?

MULTISECANT METHOD

- Quasi-Newton methods are based on enforcing the Secant condition on every new iterate/gradient update

$$W_{k+1} = \begin{cases} \underset{W}{\text{minimize}} & \|W - W_k\| \\ \text{subject to} & \boxed{W\Delta g_k = \Delta x_k} \end{cases} \qquad \begin{aligned} W &= \left(\nabla^2_{xx} f(x)\right)^{-1} \\ g &= \nabla_x f(x) \end{aligned}$$

- Multisecant methods construct an approximation by simultaneously enforcing the Secant condition on "q" stored iterates where $\widetilde{W}$ is a "preconditioner"

$$W_k = \begin{cases} \underset{H}{\text{minimize}} & \|W - \tilde{W}\| \\ \text{subject to} & \boxed{WG_k = X_k} \end{cases} \qquad \begin{aligned} X_k &= \begin{bmatrix} \Delta x_{k-q} & \Delta x_{k-q+1} & \dots & \Delta x_{k-1} & \Delta x_k \end{bmatrix} \\ G_k &= \begin{bmatrix} \Delta g_{k-q} & \Delta g_{k-q+1} & \dots & \Delta g_{k-1} & \Delta g_k \end{bmatrix} \end{aligned}$$

MULTISECANT METHOD

$$W_k = \begin{cases} \underset{H}{\text{minimize}} & \|W - \tilde{W}\| \\ \text{subject to} & WG_k = X_k \end{cases}$$

$$X_k = \begin{bmatrix} \Delta x_{k-q} & \Delta x_{k-q+1} & \dots & \Delta x_{k-1} & \Delta x_k \end{bmatrix}$$
$$G_k = \begin{bmatrix} \Delta g_{k-q} & \Delta g_{k-q+1} & \dots & \Delta g_{k-1} & \Delta g_k \end{bmatrix}$$

- First appearance in material sciences (Vanderbilt and Louie, 1984)
- Formally defined as a “generalized Broyden’s method” equivalent to Anderson mixing with $\widetilde{W} = \alpha I$ (Eyert, 2006)
- Shown to be effective for solving **noisy** nonlinear systems of equations (Bierlaire and Crittin, 2006)
- Adapted to PDE-constrained optimization with inaccurate forward and adjoint solves (Hicken et. al., 2017)

MULTISECANT FOR MACHINE LEARNING?

$$\underset{x}{\text{minimize}} \quad \hat{\mathcal{J}}(x;y) = \frac{1}{N_b}\sum_{i}^{N_b} \|\mathcal{M}(x;y_i) - \mathcal{R}(y_i)\|_2^2$$

$$\text{subject to} \quad \hat{\mathcal{C}}(\mathcal{M}(x;y)) = \frac{1}{N_b}\sum_{i}^{N_b} \mathcal{C}(\mathcal{M}(x;y_i)) = 0$$

Formulate the Lagrangian: $\mathcal{L} = \hat{\mathcal{J}} + \lambda^T \hat{\mathcal{C}}$

Differentiate for first-order optimality conditions:

$$\left(\begin{array}{r} \nabla_x \mathcal{L} = \nabla_x \hat{\mathcal{J}} + \lambda^T \nabla_x \hat{\mathcal{C}} = 0 \\ \nabla_\lambda \mathcal{L} = \hat{\mathcal{C}} = 0 \end{array} \right)$$

This is a "noisy" system of nonlinear equations!

MAD: MULTISECANT ACCELERATED DESCENT

Find (x^*, λ^*) such that: $\begin{bmatrix} \nabla_x \mathcal{L} \\ \nabla_\lambda \mathcal{L} \end{bmatrix} = \begin{bmatrix} \nabla_x \hat{\mathcal{J}}(x^*) + {\lambda^*}^T \nabla_x \hat{\mathcal{C}}(x^*) \\ \hat{\mathcal{C}}(x^*) \end{bmatrix} = \mathbf{0}$

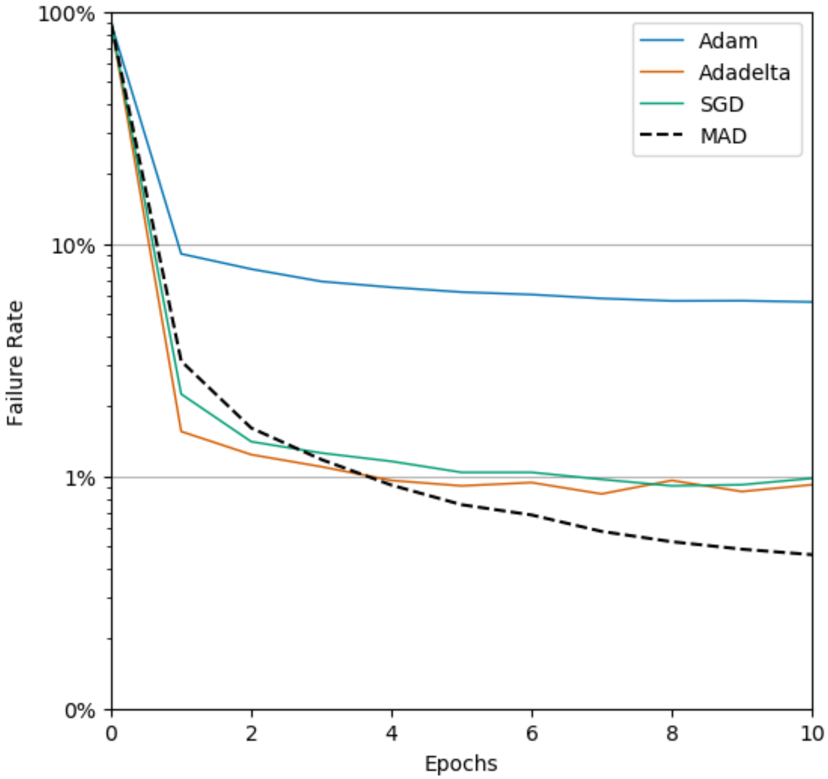
Set $\hat{x}_0 = [x_0, \lambda_0]$ and $g_0 = [\nabla_x \mathcal{L}_0, \nabla_\lambda \mathcal{L}_0]$
Take a gradient descent step $p_1 = p_0 - \eta g_0$
for $k = 1, 2, \ldots$ **do**
  Update X_k and G_k with $(\hat{x}_k - \hat{x}_{k-1})$ and $(g_k - g_{k-1})$
  Solve least squares problem $\gamma = \arg\min_\gamma \|g_k - G_k \gamma\|$
  Compute step direction $\Delta \hat{x}_k = -\tilde{W} g_k - (X_k - \tilde{W} G_k)\gamma$
  Update $\hat{x}_{k+1} = \hat{x}_k + \eta \Delta \hat{x}_k$
end for

TEST CASE: IMAGE CLASSIFICATION

Image source: Josef Steppan (wikimedia.org)

- Handwritten digit classification problem
- 60,000 images for training and 10,000 for testing
- Cloned from pyTorch MNIST example
    - Two Conv2D layers
    - Two fully-connected linear layers
    - Softmax output layer
    - 1,199,882 parameters

TEST CASE: MNIST CLASSIFICATION



	Adam	Adadelta	SGD
Init. LR	1.0	N/A	0.001

- “Reduce LR on plateau” scheduler
- MAD Parameters:

$$q = 10 \qquad \eta_0 = 0.01$$

- pyTorch defaults for others

TEST CASE: XGC FPL COLLISION OPERATOR

- **XGC** is a hybrid Lagrangian-Eulerian particle-in-cell based gyrokinetic code used for simulating the edge region of fusion devices

- Fokker-Planck-Landau (FPL) collision operator in XGC scales quadratically with number of species

Can we accelerate XGC by replacing collision operator with a DNN?

DNN must conserve mass, momentum and energy!

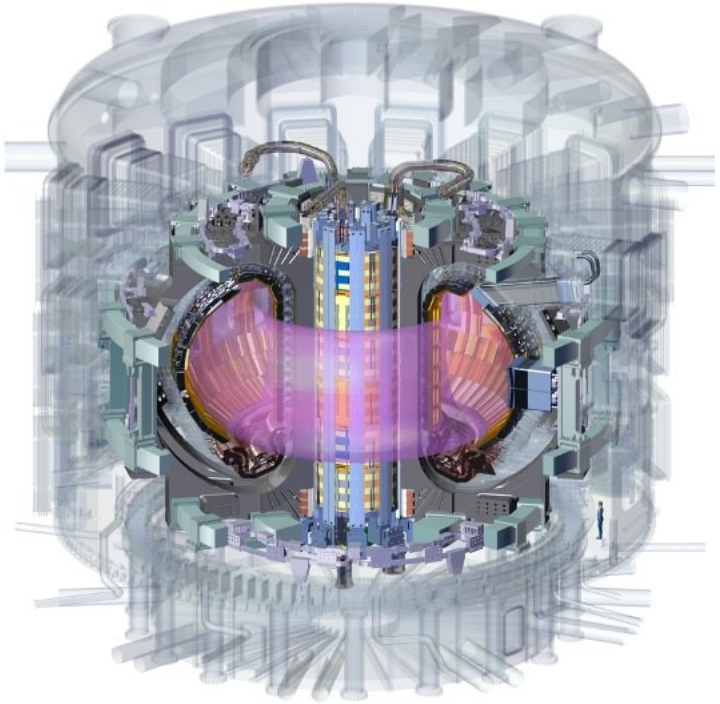
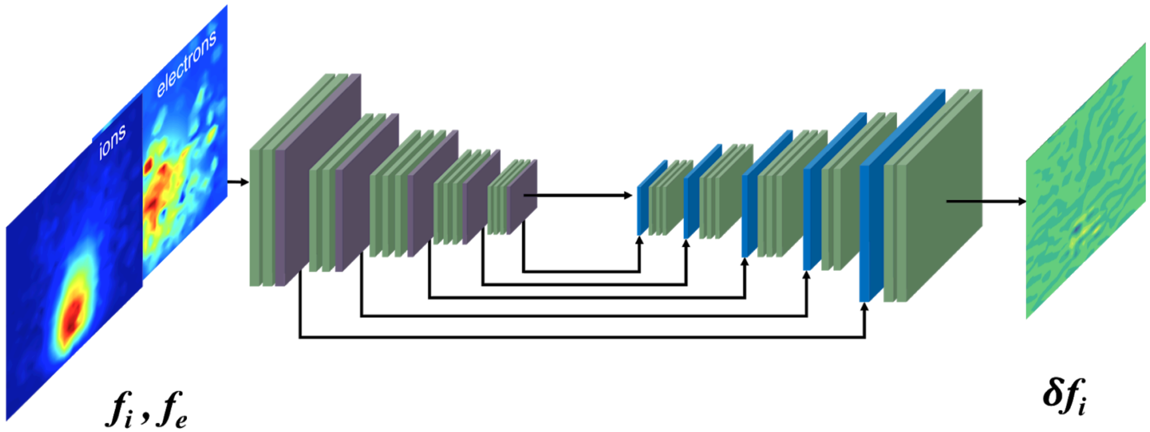


Image Source: iter.org

TEST CASE: XGC FPL COLLISION OPERATOR

- Collision operator computes the change in velocity distribution field of one species (Δf_i) based on the current velocity distributions of all species $(f_i; f_e)$
- Replace FPL collision operator with a ReSeg network (Visin et. al. 2016)

$$\Delta f_i^{ML} = \mathcal{M}(p; f_i^{XGC}; f_e^{XGC})$$



- 2,664,877 parameters
- 96,000 data points for training
- 12,000 data points for validation
- 12,000 data points for out-of-sample testing

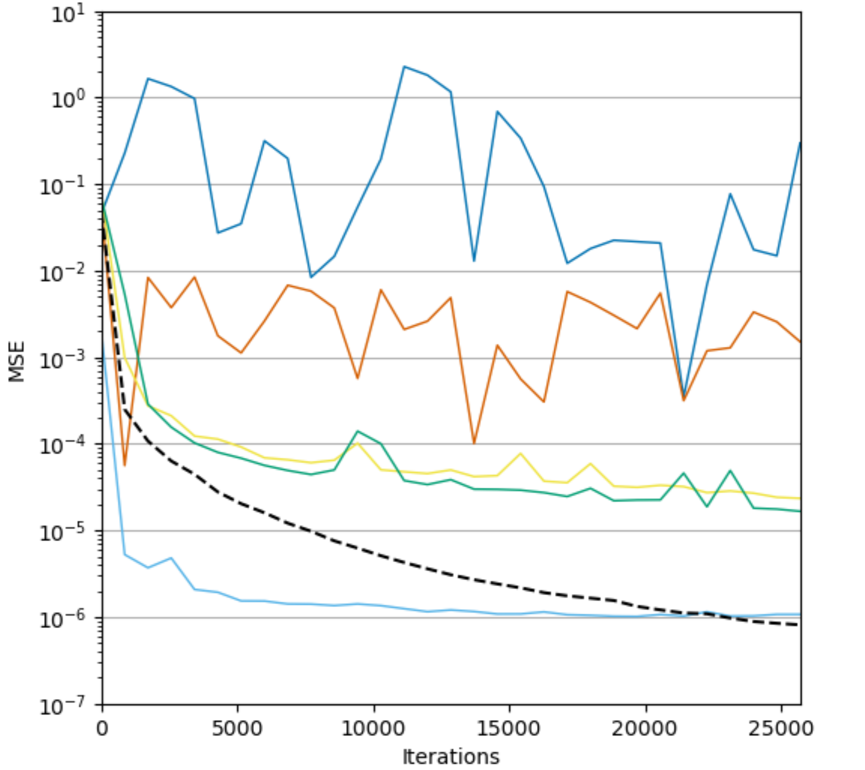
TEST CASE: XGC FPL COLLISION OPERATOR

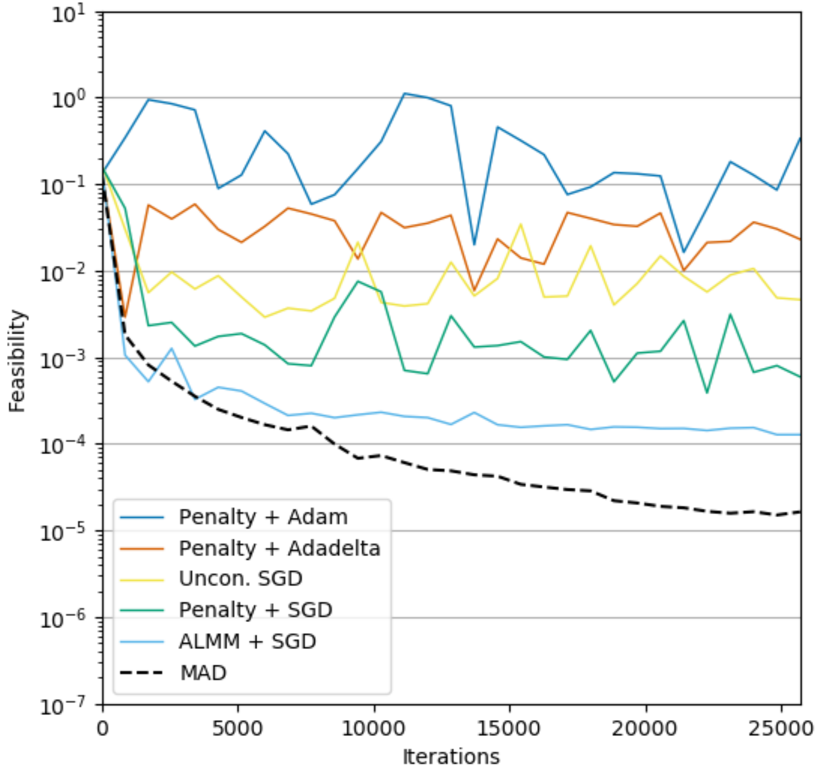
$$\underset{p}{\text{minimize}} \quad \hat{\mathcal{J}}(p) = \frac{1}{2N}\sum_{j}^{N} \|\mathcal{M}(p; f_{i,j}^{XGC}; f_{e,j}^{XGC}) - \Delta f_{i,j}^{XGC}\|_2^2$$

$$\text{subject to} \quad \hat{\mathcal{C}}(p) = \frac{1}{N}\sum_{j}^{N} \begin{pmatrix} \Delta m(\mathcal{M}(p; f_{i,j}^{XGC}; f_{e,j}^{XGC})) \\ \Delta P(\mathcal{M}(p; f_{i,j}^{XGC}; f_{e,j}^{XGC})) \\ \Delta E(\mathcal{M}(p; f_{i,j}^{XGC}; f_{e,j}^{XGC})) \end{pmatrix} = \mathbf{0}$$

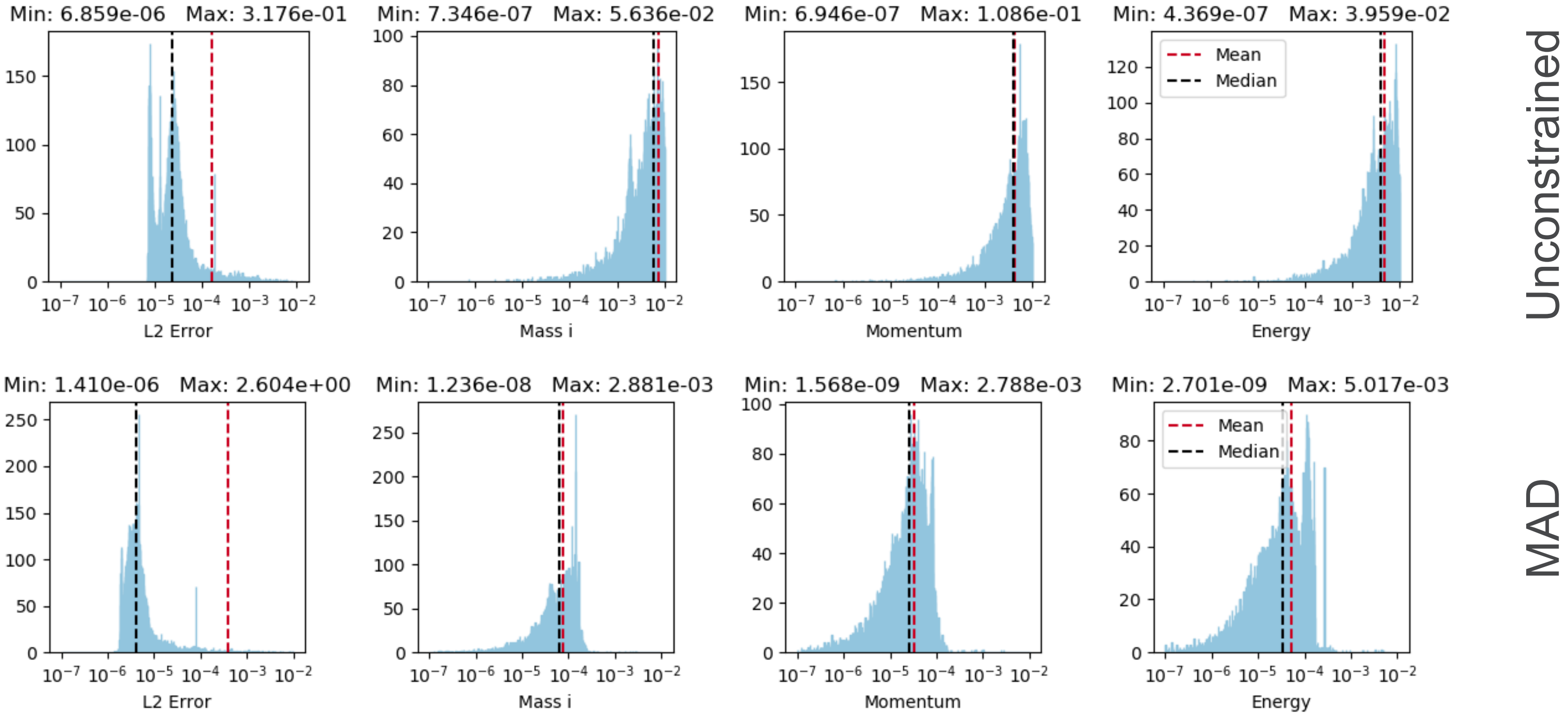
- Batch-averaged conservation constraints on mass (m), momentum (P) and energy (E)
- In previous work, we solved this with a stochastic augmented-Lagrangian method of multipliers (Dener et. al., 2020) – see Todd’s talk in MS80
- In current work, we try a different approach with MAD and compare to SALMM as well as a penalty method

TEST CASE: XGC FPL COLLISION OPERATOR





TEST CASE: XGC FPL COLLISION OPERATOR



WRAPPING UP…

What did we learn?

- We can and should do better than just penalizing the loss function!
- PINN training methods need to be guided by first-order optimality conditions
- No such thing as parameter-free training, but maybe we can trade hard-to-tune parameters (i.e. the penalty factor) for more benign ones
- Most of this is “old ideas made new again”

Where do we go from here?

- Quasi-Newton Hessian initialization techniques for ”preconditioner” $\widetilde{W}$
- Efficient ways to adapt non-monotone line searches to ML training
- More to learn from literature on solving noisy nonlinear systems of equations
- Visualizing the *constrained* loss landscape (Li et. al., 2018)

RELATED WORKS

Miller, M. A., Churchill, R. M., Dener, A., Chang, C. S., Munson, T., & Hager, R "*Encoder-decoder neural network for solving the nonlinear Fokker-Planck-Landau collision operator in XGC*" arXiv preprint arXiv:2009.06534 (2020)

Dener, A., Miller, M. A., Churchill, R. M., Munson, T., & Chang, C. S. "*Training neural networks under physical constraints using a stochastic augmented Lagrangian approach*" arXiv preprint arXiv:2009.07330 (2020)

Hicken, J. E., Meng, P., & Dener, A. "*Error-tolerant multisecant method for nonlinearly constrained optimization*" arXiv preprint arXiv:1709.06985 (2017)

This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program via the FASTMath Institute under Contract No. DE-AC02-06CH11357 at Argonne National Laboratory and via the Partnership Center for High-fidelity Boundary Plasma Simulation at Princeton Plasma Physics Laboratory under the Contract No. DE-AC02–09CH11466.

The XGC simulations used computational resources at the Argonne (Theta) and Oak Ridge (Summit) Leadership Computing Facilities, DOE Office of Science User Facilities supported under Contracts DE-AC02-06CH11357 and DE-AC05-00OR22725, respectively.