MN Method Notes

Update 07/22/20

Contents

1 The M$_N$ Closure Framework

1.1 Entropy Maximization

We define a kinetic distribution function $f = f(x, \mu, t)$ for $x \in \mathbb{R}, \mu \in V = [-1, 1], t \geq 0$ which evolves according to the following **linear kinetic equation**

$$\partial_t f + \mu \partial_x f + \sigma_a f = \sigma_s(\frac{1}{2}\langle f \rangle - f) \tag{1.1.1}$$

Let $\mathbf{m_N} \colon V \longrightarrow \mathbb{R}^{N+1}$ be the vector of Legendre polynomials on the velocity domain up to degree N (including the 0th degree). We can integrate (**??**) against $\mathbf{m_N}$, defining $\mathbf{u}_N := \langle \mathbf{m}_N \cdot f \rangle$, to obtain:

$$\partial_t \mathbf{u}_N + \partial_x \langle \mu \mathbf{m}_N f \rangle + \sigma_a \mathbf{u}_N = -\sigma_s \mathbf{Q} \mathbf{u}_N$$

This is not "closed" in the first $N+1$ moments of f, since the second term on the lefthand-side contains higher order moments of f. We close this as an evolution of $\mathbf{u}_N$ by approximating f as some function of the moments $G(\mathbf{u}_N)(x, t)$, to obtain

$$\partial_t \mathbf{u}_N + \partial_x \langle \mu \mathbf{m}_N G(\mathbf{u}_N) \rangle + \sigma_a \mathbf{u}_N = -\sigma_s \mathbf{Q} \mathbf{u}_N \tag{1.1.2}$$

The MN closure framework is defined by the following minimization problem:

Definition (M_N System)**.** *In (**??**), define the closure $G(\mathbf{u}_N)$ as the extremal value of the following minimization (entropy-maximization) problem*

$$\begin{aligned} &\textit{mininimize } \langle \eta(g) \rangle, \; g \in L^1(V) \\ &\textit{subject to } \langle \mathbf{m}_N g \rangle = \mathbf{u}_N \end{aligned} \tag{1.1.3}$$

where $\langle \cdot \rangle$ denotes integration with respect to the velocity domain V, and $\eta(r) = r\log(r) - r$; for the case that $\dim(V) = 1$, $\mathbf{m}$ is the vector of up-to-$N+1$ order Legendre polynomials on the velocity domain.

Definition (Minimum Entropy)**.**

$$h(\mathbf{u}) = \min \langle \eta(g) \rangle \textit{ subject to } g \in L^1(V), \; \langle \mathbf{m} g \rangle = \mathbf{u} \tag{1.1.4}$$

1.2 The Dual Problem to Entropy-Maximization

The defining minimization problem of the M$_N$ system has the following unconstrained dual problem (writing η_* for the legendre-transform of η):

$$\text{minimize } \langle \eta_*(\alpha^t \mathbf{m}) \rangle - \alpha^t \cdot \mathbf{u} \;, \alpha \in \mathbb{R}^{N+1} \tag{1.2.1}$$

where $h_*(\alpha) = \langle \eta_*(\alpha^t \mathbf{m}) \rangle$.

Definition (Dual Solution)**.** *For an admissable moment vector u*

$$\hat{\alpha}(\mathbf{u}) := \textit{argmin}\ \langle \eta_*(\alpha^t \mathbf{m})\rangle - \alpha^t \cdot \mathbf{u}\ , \alpha \in \mathbb{R}^{N+1}$$

Fact (Primal Solution)**.** *By necessary conditions for duality, the solution* $\hat{\alpha}(\mathbf{u})$ *to the dual problem obtains the primal solution to* (**??**)*. Explicitly, the function* $G_\alpha(v)$ *defined as*

$$G_\alpha(v) = \eta_*'(\alpha^t \mathbf{m}) = \exp(\alpha^t \mathbf{m}) \tag{1.2.2}$$

is the solution to (**??**)

Fact (Gradient is Dual Optimum)**.** *First order conditions for duality show that*

$$\nabla h(\mathbf{u}) = \hat{\alpha}(\mathbf{u}) \tag{1.2.3}$$

From the above fact, it immediately follows that $\hat{\alpha}$ is therefore an invertible function, let its inverse be given by $\hat{u}(\alpha)$ for admissable vectors $\alpha \in \mathbb{R}^{N+1}$.

2 The Problem: Computing the Closure for M_1

We will restrict to the case $N = 1$ and consider

$$V = [-1, 1],\ \mathbf{m} = \begin{pmatrix} 1 \\ v \end{pmatrix}$$

Note that the functions in $\mathbf{m}$ are not L^2 normalized. In the following, the approximation to the entropy function h, and its gradient $\hat{\alpha}$, will be written as $\tilde{h}$ and $\tilde{\alpha}$.

2.1 The Problem

To numerically solve a discretization of the PDE (**??**) via a Runge-Kutta scheme, we must compute $G(\mathbf{u}_N)$ via $G_\alpha(v)$, which is defined by (**??**). Each update to the numerical $\mathbf{u}_N$, requires the computation of $\hat{\alpha}(\mathbf{u})$. This can be done accurately for each value of $\mathbf{u}_N$, for example, via a gradient-descent optimization algorithm. This method has the advantage that, with high accuracy for values of α, we are assured that $\hat{\alpha}(\mathbf{u})$ are inherited from the convex function h, ensuring our solution enjoys a variety of desireable properties. However, this is computationally slow – the optimization routine becomes much slower as the number of moments (N) increases, and similarly, if the velocity and position domains increase in dimension.

2.2 Proposed Method

We can approximate the entropy function h with a function $\tilde{h}$, which is convex and C^2, but which also admits a closed-form, forward-evaluation expression for its gradient $\hat{\alpha}$.

1. We approximate the entropy function $h\colon \mathcal{R} \longrightarrow \mathbb{R}$ with $\tilde{h}$ using
   (a) A fully-connected artificial Neural-Network
   (b) Rational-Cubic Spline
2. Compute $\nabla\tilde{h}(\mathbf{u})$ and define $\tilde{\alpha} := \nabla\tilde{h}(\mathbf{u})$

In principal, this can be done on the full 2-dimensional realizable set for the M$_1$*equation*. However, in practice, it yielded much better results to approximate h on a 1-dimensional segment, and scale the results to the full 2-dimensional realizable set.

2.3 Metric for Approximation Accuracy

The term which we desire to compute more efficiently in (**??**) is the so-called "flux-update", i.e.

$$F\colon \mathcal{R} \longrightarrow \mathbb{R}^2$$

$$F(\mathbf{u}) := \langle \mu \mathbf{m}_N G_{\hat{\alpha}(\mathbf{u})} \rangle$$

Fact. *It follows from calculus that*

$$\|F(\mathbf{u}) - F(\mathbf{v})\| \leq C\|\mathbf{u} - \mathbf{v}\|$$

For an approximate closure $\tilde{h}, \tilde{\alpha}$, u and v are replaced in this estimate $\hat{u}(\tilde{\alpha}(u))$ and by $\hat{u}(\tilde{\alpha}(v))$ - the moment-reconstruction induced by our approximation to α. Since the relationship between $\tilde{\alpha}$ and $\hat{u}(\tilde{\alpha})$ is defined by the integral of an exponential, we consider the $\hat{u}(\tilde{\alpha}(u))$ to be the best metric of our accuracy.

For accuracy, we intend to obtain an approximation which minimizes the value

$$\|\hat{u}(\tilde{\alpha}(u)) - u\|_{L^2(\mathcal{R})}$$

3 Dimension-Reduction: Infinite Cone to Bounded Segment

3.1 Scaling Relations

From the definitions in Section 1, it is straightforward to show that

Fact (Alpha Scaling)**.**

$$\forall \lambda > 0$$

$$\hat{\alpha}(\lambda \mathbf{u}) = \alpha(\mathbf{u}) + \begin{pmatrix} \ln(\lambda) \\ 0 \end{pmatrix} \tag{3.1.1}$$

from which it follows that

Fact (Entropy Scaling)**.**

$$\forall\ \lambda > 0$$

$$h(\lambda \mathbf{u}) = \lambda(h(\mathbf{u}) + u_0 \ln(\lambda)) \tag{3.1.2}$$

For the case of the M_1 equation, where the moment vectors are simply $\mathbf{u} = (u_0, u_1)$, this gives the following:

Fact (u_0 Scaling)**.** *For $v \in \mathcal{R}$, choose $\lambda = v_0$ and $\mathbf{v} = v_0 \begin{pmatrix} 1 \\ \frac{v_1}{v_0} \end{pmatrix}$*

$$h(\mathbf{v}) = v_0(h((\frac{\mathbf{v}}{v_0})) + ln(v_0)) \tag{3.1.3}$$

To approximate h, we can approximate the behavior of h on the set $\mathcal{D} = \{v \in \mathcal{R} \colon v_0 \equiv 1\}$, and extend the network to $\mathcal{R}$ using this scaling relation. That is

Definition. *We write $\tilde{h}_s$ for a function*

$$\tilde{h}_s \colon \mathcal{D} \longrightarrow \mathbb{R}$$

which aproximates the entropy function h on the set $\mathcal{D}$

$$\tilde{h}\colon \mathcal{R} \longrightarrow \mathbb{R}$$

$$\tilde{h}(\mathbf{u}) := u_0 \cdot \tilde{h}_s((1, u_1/u_0)) + u_0 \cdot \ln(u_0)$$

3.2 Guaranteeing Convexity

It is not immediately obvious (to me) that an approximation $\tilde{h}_s$ which is convex will induce a convex function $\tilde{h}$. This, however, turns out to be the case; we prove this by checking the positive-definiteness of the hessian matrix for $\tilde{h}$.

4 The Network Approximation

4.1 Defining Map

Using the approximation suggested by (**??**), and again recalling $D := \{u \in \mathcal{R}\colon u_0 \equiv 1\}$ we define $\mathcal{N}\colon \mathcal{D} \longrightarrow \mathbb{R}$ to be a neural network approximating h, and extend the approximation defined by $\mathcal{N}$ to all of $\mathcal{R}$ as $\tilde{h}$ by:

Definition. *(Network Approximation) Define* $\tilde{h}\colon \mathcal{R} \longrightarrow \mathbb{R}$ *via*

$$\begin{aligned} \tilde{h}(\mathbf{v}) &= \frac{v_1}{v_0} * (\mathcal{N}(\frac{v_1}{v_0}) + \log v_0) \\ \xi(v) &:= v_0 \mathcal{N}(\frac{v_1}{v_0}) \text{ and } f(v) := v_0 \log v_0 \end{aligned} \tag{4.1.1}$$

In short, this means that we will only train the network on the set

$$D = \{u \in \mathcal{R}\colon u_0 \equiv 1\} = \{(1, v)\colon |v| < 1\}$$

and extend it to the rest of the space by the scaling relation above.

Given a network $\mathcal{N}$ and it's gradient $\mathcal{N}'$ at the set $u_0 \equiv 1$ which approximates h, equations (**??**) and (**??**) induce two separate ways to extend the approximate value of α, called $\tilde{\alpha}$, to the full set $\mathcal{R}$.

1. The first, using equation (**??**) is:

$$\tilde{\alpha}(\mathbf{v}) := \tilde{\alpha}(\mathbf{v}/v_0) + \begin{pmatrix} \log v_0 \\ 0 \end{pmatrix}$$

2. The second, using (**??**) is:

$$\tilde{\alpha}(\mathbf{v}) := \nabla \tilde{h}(\mathbf{v}) = \nabla f(\mathbf{v}) + \nabla \xi(\mathbf{v})$$

   since both f and ξ have global definitions.

In the second case, this gives

$$\nabla f(v) = \begin{pmatrix} 1 + \ln(v_0) \\ 0 \end{pmatrix}$$

and

$$\nabla \xi(v) = \begin{pmatrix} \mathcal{N}(\frac{v_1}{v_0}) - \frac{v_1}{v_0} \cdot \mathcal{N}'(\frac{v_1}{v_0}) \\ \frac{1}{v_0} \mathcal{N}'(\frac{v_1}{v_0}) \end{pmatrix}$$

In total this gives

$$\tilde{\alpha}(\mathbf{v}) := \nabla f(\mathbf{v}) + \nabla \xi(\mathbf{v}) = \begin{pmatrix} 1 + \ln(v_0) + \mathcal{N}(\frac{v_1}{v_0}) - \frac{v_1}{v_0} \cdot \mathcal{N}'(\frac{v_1}{v_0}) \\ \frac{1}{v_0}\mathcal{N}'(\frac{v_1}{v_0}) \end{pmatrix} \tag{4.1.2}$$

In either case, we have that restricting to the domain of definition of $\mathcal{N}$, i.e. setting $v_0 \equiv 1$, gives

$$\tilde{\alpha}((1, v_1)) = \begin{pmatrix} 1 + \mathcal{N}(v_1) - v_1 \cdot \mathcal{N}'(v_1) \\ N'(v_1) \end{pmatrix} \tag{4.1.3}$$

In either regime, we always define

$$\tilde{u}(\mathbf{v}) := \hat{u}(\tilde{\alpha}(\mathbf{v})) \tag{4.1.4}$$

In the case of the networks that follow, we will use the first definition for extending $\tilde{\alpha}$. This ensures that the global error in the approximation, $\tilde{\alpha}$, is identical to the $\tilde{\alpha}$ error at the $u_0 \equiv 1$ line.

4.2 Network Results

4.3 Network 1d19: (5,40) Architecture, Early Stopping at 4366 / 8000 Epochs

Patience was 200 epochs, learning-rate drops 1 order of magnitude per 2000 epochs.

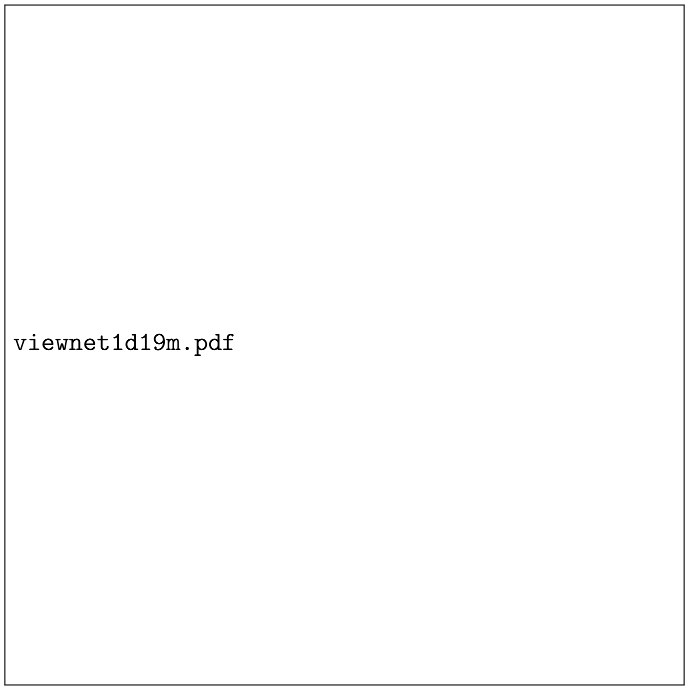


Figure 1: Network Results Training Data

	f	r-f	r-alpha	u	r-u	# NegDef
0	7.75e-07	4.18e-07	7.43e-07	2.41e-06	1.30e-06	0

Table 1: MSE and Conv on Training Data

losscurve1d19m-eps-converted-to.pdf

complosscurvetrain1d19m-eps-converted-to.pdf

Figure 2: Loss Curve on Training Data

complosscurveval1d19m-eps-converted-to.pdf

Figure 3: Loss Curve on Validation Data

	h	r-h	u	r-u	alpha	r-alpha
0	2.32e-05	1.76e-06	4.65e-06	1.47e-06	1.10e-03	5.54e-07

Table 2: MSE Over Test Domain

	MSE alpha-0	MSE alpha-1	r-MSE alpha-0	r-MSE alpha-1	rel-sq alpha-0	rel-sq alpha-1
0	5.16e-04	5.83e-04	2.60e-07	2.94e-07	3.88e-03	3.19e-05

Table 3: Alpha Error Over Test Domain

alphaheat1d19m-eps-converted-to.pdf

alpha0heat1d19m-eps-converted-to.pdf

alpha1heat1d19m-eps-converted-to.pdf

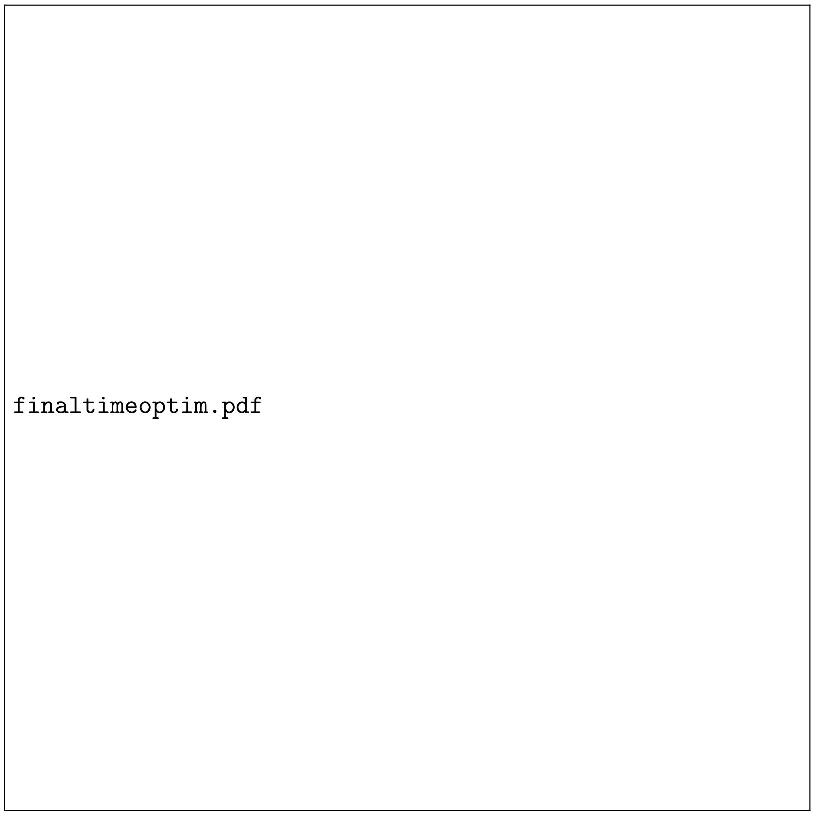
Figure 4: Test-Domain Alpha Error Values (in norm)

funcheat1d19m-eps-converted-to.pdf

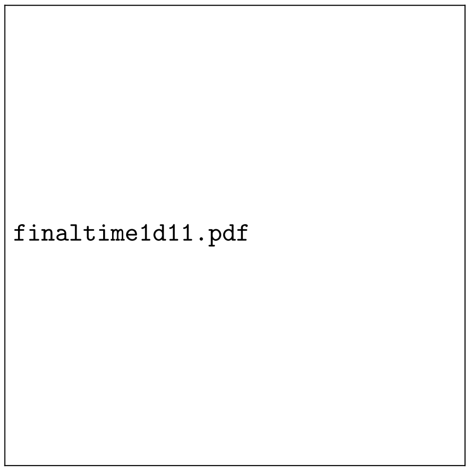
momentheat1d19m-eps-converted-to.pdf

momentu0heat1d19m-eps-converted-to.pdf

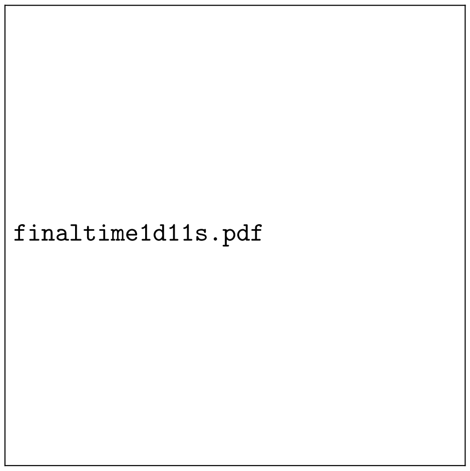
5 Numerical Solutions: Plane Test



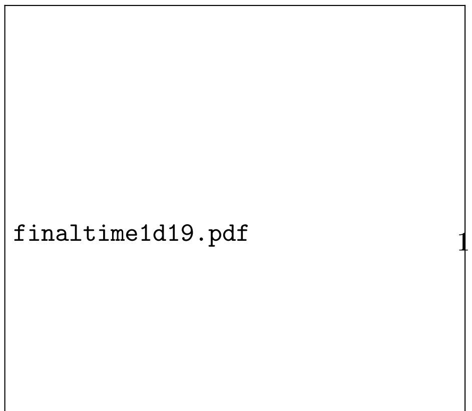
(a) MN-Optim

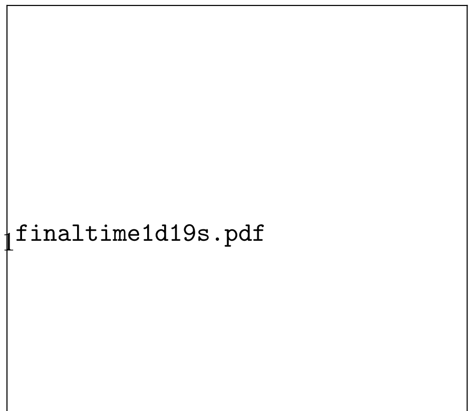


(b) Net "1d11"-20 x 80



(c) Net"1d11"- 20x80 Symm.





	MN-Optim	1d19 Network	1d11 Network
Non-Symmetrized	30.37	17.10	18.66
Symmetrized	N/A	40.15	50.52

Table 4: Computation Time With Plot Feature (in seconds)