Predicting Flow Reversals in a Computational Fluid Dynamics Simulated Thermosyphon using Data Assimilation

Andrew Reagan,[1] Yves Dubief,[2] Peter Sheridan Dodds,[1] and Christopher M. Danforth[1]

[1]*Department of Mathematics & Statistics, Vermont Complex Systems Center, Computational Story Lab, & the Vermont Advanced Computing Core, The University of Vermont, Burlington, VT 05401*

[2]*School of Engineering, Vermont Complex Systems Center & the Vermont Advanced Computing Core, The University of Vermont, Burlington, VT 05401*

(Dated: October 2, 2015)

A thermal convection loop is a circular chamber filled with water, heated on the bottom half and cooled on the top half. With sufficiently large forcing of heat, the direction of fluid flow in the loop oscillates chaotically, forming an analog to the Earth's weather. As is the case for state-of-the-art weather models, we only observe the statistics over a small region of state space, making prediction difficult. To overcome this challenge, data assimilation (DA) methods, and specifically ensemble methods, use the computational model itself to estimate the uncertainty of the model to optimally combine these observations into an initial condition for predicting the future state. Here, we build and verify four distinct DA method, and then, we perform a twin model experiment with the computational fluid dynamics simulation of the loop using the Ensemble Transform Kalman Filter (ETKF) to assimilate observations and predict flow reversals. We show that using adaptively shaped localized covariance outperforms static localized covariance with the ETKF, and allows for the use of less observations in predicting flow reversals. We also show that a Dynamic Mode Decomposition (DMD) of the temperature and velocity fields recovers the low dimensional system underlying reversals, finding specific modes which together are predictive of reversal direction.

PACS numbers:

I. INTRODUCTION

Prediction of the future state of complex systems is a fundamental challenge of science and engineering, and ultimately integral to the functioning of society. Some of these systems include weather [18], health [13], the economy [35], marketing [2] and transportation [34]. For weather in particular, predictions are made using supercomputers integrating numerical weather models, projecting our current best guess of the weather into the future. The accuracy of these predictions depends on the accuracy of the models themselves, and the quality of our knowledge of the current state of the atmosphere.

Model accuracy has improved with better meteorological understanding of weather processes and advances in computing technology [3]. To solve the initial value problem, techniques developed over the past 50 years are now broadly known as *data assimilation* (DA). Formally, data assimilation is the process of using all available information, including short-range model forecasts and physical observations, to estimate the current state of a system as accurately as possible [39]. The best-guess of the current state is often referred to as the *analysis* state.

Here, we employ a fluid dynamics experiment as a test bed for improving numerical weather prediction algorithms, focusing specifically on data assimilation methods. Our approach is inspired by the historical development of current methodologies, and provides a tractable system for rigorous analysis. The experiment is a thermal convection loop, which by design simplifies our problem into the prediction of convection. The thermosyphon, a type of natural convection loop or non-mechanical heat pump, can be likened to a toy model of climate [17]. The dynamics of thermal convection loops have been explored under both periodic [25] and chaotic [6, 7, 9, 10, 15, 16, 22, 33, 38–40] regimes. A full characterization of the computational behavior of a loop under flux boundary conditions by Louisos et. al. describes four regimes: chaotic convection with reversals, high Rayleigh number (Ra) aperiodic stable convection, steady stable convection, and conduction/quasi-conduction [28]. For the remainder of this work, we focus on the chaotic flow regime.

We perform all computational simulations of the thermal convection loop with the open-source finite volume C++ library OpenFOAM [21]. The open-source nature of this software enables its integration with the data assimilation framework that our present work provides.

II. PHYSICAL EXPERIMENT AND COMPUTATIONAL MODEL

The reduced order system describing a thermal convection loop was originally derived by Gorman [16] and Ehrhard and Müller [10]. Here we present this three dimensional system in non-dimensionalized form. In Appendix B, we present a more complete derivation of these equations, following the derivation of Harris [17]. For the mean fluid velocity $\frac{\mathrm{d}x_1}{\mathrm{d}t}$, temperature difference between the 3 o'clock and 9 o'clock positions $\frac{\mathrm{d}x_2}{\mathrm{d}t}$ (also referred to presently as ΔT_{3-9}), and deviation from con-

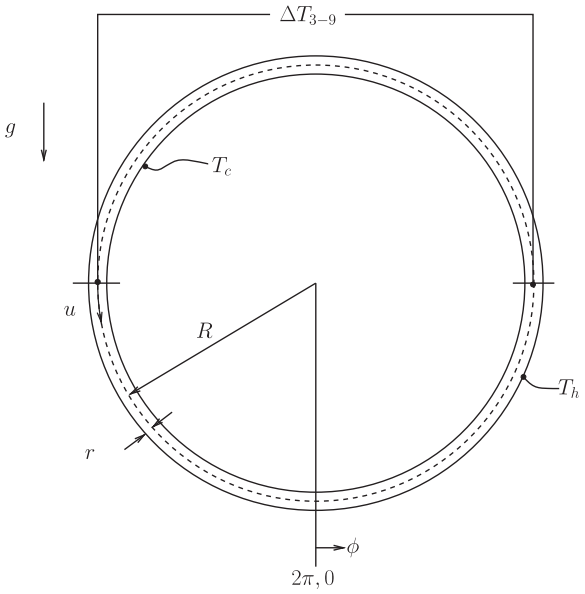

FIG. 1: Schematic of the experimental, and computational, setup from Harris *et al.*(2012). The loop radius is given by R and inner radius by r. The top temperature is labeled T_c and bottom temperature T_h, gravity g is defined downward, the angle ϕ is prescribed from the 6 o'clock position, and temperature difference between 3 o'clock and 9 o'clock posititions ΔT_{3-9} is labeled.

ductive temperature profile $\frac{\mathrm{d}x_3}{\mathrm{d}t}$, these equations are:

$$\frac{\mathrm{d}x_1}{\mathrm{d}t} = \alpha(x_2 - x_1), \tag{1}$$

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} = \beta x_1 - x_2(1 + Kh(|x_1|)) - x_1 x_3, \tag{2}$$

$$\frac{\mathrm{d}x_3}{\mathrm{d}t} = x_1 x_2 - x_3(1 + Kh(|x_1|)). \tag{3}$$

The function $h(x)$ is a defined piecewise analytic polynomial, and is provided in the full derivation as Equation B11. The parameters α, β, and K, along with scaling factors for time and each model variable can be fit to data using standard parameter estimation techniques.

Operated by Dave Hammond, UVM's Scientific Electronics Technician, the experimental thermosyphons access the chaotic regime of state space found in the principled governing equations. We quote the detailed setup from Darcy Glenn's undergraduate thesis [14] and provide Figure 1 for details of the experiment:

> The [thermosyphon] is a bent semi-flexible plastic tube with a 10-foot heating rope wrapped around the bottom half of the upright circle. The tubing used is light-transmitting clear THV from McMaster-Carr, with an inner diameter of 7/8 inch, a wall thickness of 1/16 inch, and a maximum operating temperature of 200F. The outer diameter of the circular thermosyphon is 32.25 inches. Together, the tubing inner diameter and outer diameter of the thermosyphon produce a ratio of approximately 1:36. There are 1 inch 'windows' when the heating cable is coiled in a helix pattern around the outside of the tube, so the heating is not exactly uniform. The bottom half is then insulated using aluminum foil, which allowed fluid in the bottom half to reach 176F. A forcing of 57 V, or 105 Watts, is required for the heating cable so that chaotic motion is observed. Temperature is measured at the 3 o'clock and 9 o'clock positions using unsheathed copper thermocouples from Omega.

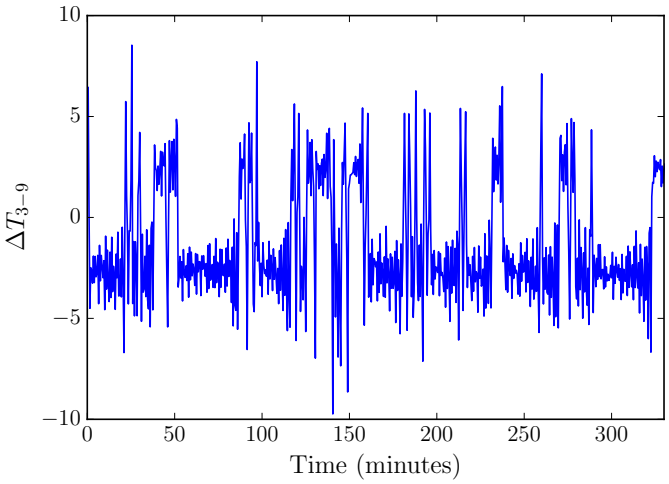

FIG. 2: A time series of the physical thermosyphon, from the Undergraduate Honor's Thesis of Darcy Glenn [14]. The temperature difference (plotted) is taken as the difference between temperature sensors in the 3 and 9 o'clock positions. The sign of the temperature difference indicates the flow direction, where positive values are clockwise flow.

We comfirm that the experiment accesses the chaotic regime of state space using a time series of the temperature difference as measured at the 3 o'clock and 9 o'clock positions in Figure 2. We first test our ability to predict this experimental thermosyphon using synthetic data.

We consider the incompressible Navier-Stokes equations with the Boussinesq approximation to model the flow of water inside a thermal convection loop. For brevity, we omit the equations themselves, and include them in the Appendix. The solver in OpenFOAM that we use, with some modification, is `buoyantBoussinesqPimpleFoam`. Solving is accomplished by the Pressure-Implicit Split Operator (PISO) algorithm [20]. We find that modification of the code is necessary for laminar operation.

We create both 2-dimensional and 3-dimensional meshes using OpenFOAM's native meshing utility `blockMesh`

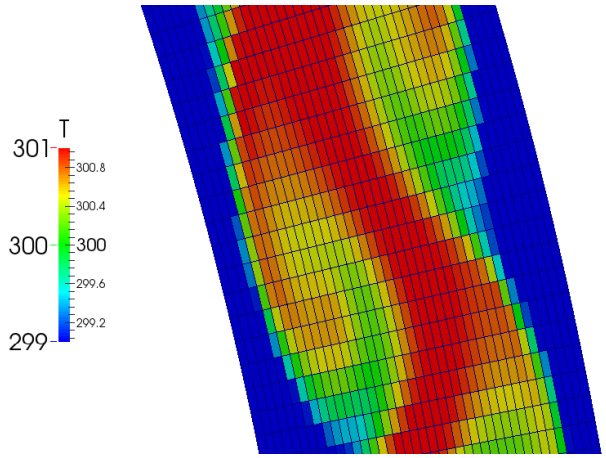

FIG. 3: A snapshot of the mesh used for CFD simulations. Shown is an initial stage of heating for a fixed value boundary condition, 2D, laminar simulation with a mesh of 40000 cells including wall refinement with walls heated at 340K on the bottom half and cooled to 290K on the top half.

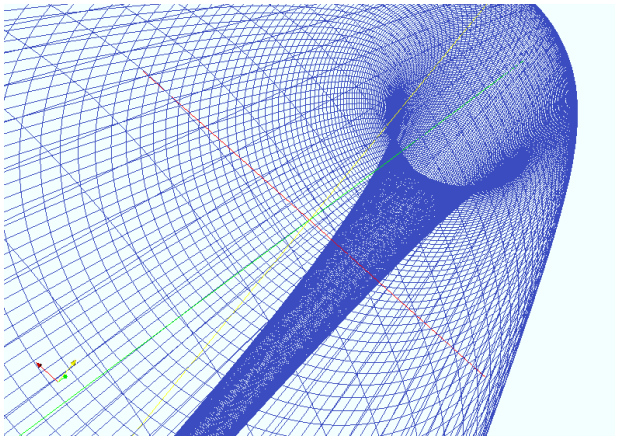

FIG. 4: The 3D mesh viewed as a wire-frame from within. Here there are 900 cells in each slice (not shown), for a total mesh size of 81,000 cells. Simulations using this computational mesh are prohibitively expensive for use in a real time ensemble forecasting system, but are possible offline.

shown in Figures 3 and 3. After creating a mesh, we refine the mesh near the walls to capture boundary layer phenomena and renumber the mesh for solving speed. We use the `refineWallMesh` utility to refine the mesh near walls, and the `renumberMesh` utility to renumber the mesh. The resulting 2D mesh contains 40,000 points (40 across the diameter and 1000 around).

Available boundary conditions (BCs) we find to be stable in OpenFOAM's solver are constant gradient, fixed value conditions, and turbulent heat flux. Simulations with a fixed flux BC is implemented through the `externalWallHeatFluxTemperature` library are unstable and resulted in physically unrealistic results. Constant gradient simulations are stable, but the behavior is empirically different from our physical system. While it is possible that a fixed value BC is acceptable due to the thermal diffusivity and thickness of the walls of the experimental setup, we find that this is also inadequate. We employ the third-party library `groovyBC` to use a gradient condition that computes a flux using a fixed external temperature and fixed wall heat transfer coefficient.

With the mesh, BCs, and solver chosen, we now simulate the flow. From the data of T, ϕ, u, v, w and p that are saved at each timestep, we extract the mass flow rate and average temperature at the $12, 3, 6$ and 9 o'clock positions on the loop. Since ϕ is saved as a face-value flux, we compute the mass flow rate over the cells i of top (12 o'clock) slice as

$$\sum_i \phi_{f(i)} \cdot v_i \cdot \rho_i \tag{4}$$

where $f(i)$ corresponds the face perpendicular to the loop angle at cell i and ρ is reconstructed from the Boussinesq approximation $\rho = \rho_{\text{ref}}(1 - \beta(T - T_{\text{ref}}))$.

III. METHODS

A. Data Assimilation

We perform tests of the data assimilation algorithms described here with the Lorenz '63 system, which is analogous to the above equations with Lorenz's $\beta = 1$, and $K = 0$. The canonical choices of $\sigma = 10, \beta = 8/3$ and $\rho = 28$ produce the well known butterfly attractor, and we use these values for all examples here. From these tests, we will find the optimal data assimilation parameters (inflation factors) for predicting time series with this system. Having done so, we then focus our efforts on making prediction using computational fluid dynamics models.

We first implement the 3D-Var filter. Simply put, 3D-Var is the variational (cost-function) approach to finding the analysis. It has been shown that 3D-var solves the same statistical problem as optimal interpolation (OI) [27]. The usefulness of the variational approach comes from the computational efficiency, when solved with an iterative method. Specifically, the multivariate 3D-Var amounts to finding the $\mathbf{x}_a$ that minimizes the cost function

$$J(\mathbf{x}) = (\mathbf{x}-\mathbf{x}_b)^T\mathbf{B}^{-1}(\mathbf{x}-\mathbf{x}_b)+(\mathbf{y}_o+H(\mathbf{x}))^T\mathbf{R}(\mathbf{y}_o-H(\mathbf{x})). \tag{5}$$

Next, we implement the "gold-standard" Extended Kalman Filter (EKF). The tangent linear model (TLM) is precisely the model (written as a matrix) that transforms a perturbation at time t to a perturbation at time $t + \Delta t$, analytically equivalent to the Jacobian of the model. Using the notation of Kalnay [23], this amounts to making a forecast with the nonlinear model M, and updating the error covariance matrix $\mathbf{P}$ with the TLM L, and adjoint model L^T:

$$\mathbf{x}^f(t_i) = M_{i-1}[\mathbf{x}^a(t_{i-1})],$$
$$\mathbf{P}^f(t_i) = L_{i-1}\mathbf{P}^a(t_{i-1})L_{i-1}^T + \mathbf{Q}(t_{i-1})$$

where $\mathbf{Q}$ is the noise covariance matrix (model error). In the experiments with Lorenz '63 presented in this section, $\mathbf{Q} = 0$ since our model is perfect. In numerical weather prediction, $\mathbf{Q}$ must be approximated, e.g., using statistical moments on the analysis increments [8, 26].

The analysis step is then written as (for H the observation operator):

$$\mathbf{x}^a(t_i) = \mathbf{x}^f(t_i) + \mathbf{K}_i\mathbf{d}_i, \tag{6}$$
$$\mathbf{P}^a(t_i) = (\mathbf{I} - \mathbf{K}_i\mathbf{H}_i)\mathbf{P}^f(t_i) \tag{7}$$

where

$$\mathbf{d}_i = \mathbf{y}_i^o - \mathbf{H}[x^f(t_i)]$$

is the innovation. We compute the Kalman gain matrix to minimize the analysis error covariance P_i^a as

$$\mathbf{K}_i = \mathbf{P}^f(t_i)\mathbf{H}_i^T[\mathbf{R}_i + \mathbf{H}_i\mathbf{P}^f(t_i)\mathbf{H}^T]^{-1}$$

where $\mathbf{R}_i$ is the observation error covariance. Since we are making observations of the truth with random normal errors of standard deviation ϵ, the observational error covariance matrix $\mathbf{R}$ is a diagonal matrix with ϵ along the diagonal. The most difficult (and most computationally expensive) part of the EKF is deriving and integrating the TLM. For this reason, the EKF is not used operationally, and later we will turn to statistical approximations of the EKF using ensembles of model forecasts. With our CFD model we have no such TLM, and we provide more detail on the TLM approachas applicable to the Lorenz '63 system in Appendix C.

The computational cost of the EKF is mitigated through the approximation of the error covariance matrix $\mathbf{P}_f$ from the model itself, without the use of a TLM. One such approach is the use of a forecast ensemble, where a collection of models (ensemble members) are used to statistically sample model error propagation. With ensemble members spanning the model analysis error space, the forecasts of these ensemble members are then used to estimate the model forecast error covariance.

The only difference between this approach and the EKF, in general, is that the forecast error covariance $\mathbf{P}^f$ is computed from the ensemble members, without the need for a tangent linear model:

$$\mathbf{P}^f \approx \frac{1}{K-2}\sum_{k\neq l}\left(\mathbf{x}_k^f - \overline{\mathbf{x}}_l^f\right)\left(\mathbf{x}_k^f - \overline{\mathbf{x}}_l^f\right)^T.$$

The ETKF introduced by Bishop is one type of square root filter, and we present it here to provide background for the formulation of the LETKF [4]. For a square root filter in general, we begin by writing the covariance matrices as the product of their matrix square roots. Because $\mathbf{P}_a$ and $\mathbf{P}_f$ are symmetric positive-definite (by definition), we can write

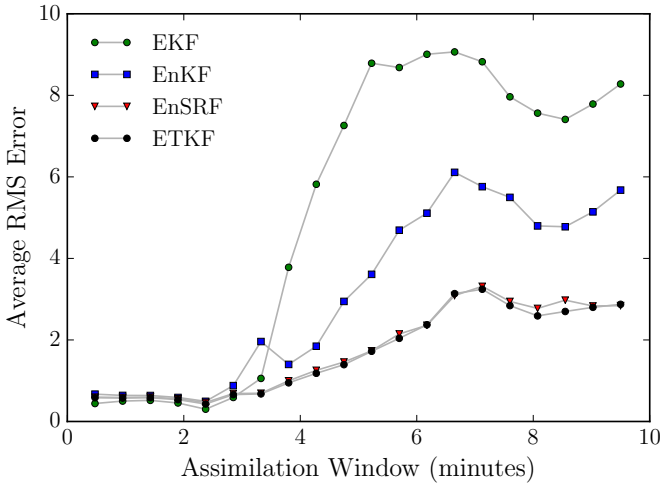

FIG. 5: The RMS error (not scaled by climatology) for our EKF and EnKF filters, measured as the difference between forecast and truth at the end of an assimiliation window for the latter 2500 assimiliation windows in a 3000 assimilation window Lorenz '63 run. Error is measured in the only observed variable, x_1. Increasing the assimilation window led to an decrease in predictive skill, as expected.

$$\mathbf{P}_a = \mathbf{Z}_a\mathbf{Z}_a^T \quad , \quad \mathbf{P}_f = \mathbf{Z}_f\mathbf{Z}_f^T \tag{8}$$

where $\mathbf{Z}_a$ and $\mathbf{Z}_f$ are the matrix square roots of $\mathbf{P}_a$ and $\mathbf{P}_f$. We are not concerned that this decomposition is not unique, and note that $\mathbf{Z}$ must have the same rank as $\mathbf{P}$ which will prove computationally advantageous. The power of the SRF is now seen as we represent the columns of the matrix $\mathbf{Z}_f$ as the difference from the ensemble members from the ensemble mean, to avoid forming the full forecast covariance matrix $\mathbf{P}_f$. The ensemble members are updated by applying the model M to the states $\mathbf{Z}_f$ such that an update is performed by

$$\mathbf{Z}_f = M\mathbf{Z}_a. \tag{9}$$

To summarize, the steps for the ETKF are to (1) form $\mathbf{Z}_f^T\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}\mathbf{Z}_f$, assuming that computing $\mathbf{R}^{-1}$ is easy, and (2) compute its eigenvalue decomposition, and apply it to $\mathbf{Z}_f$.

The LEKF implements a strategy that becomes important for large simulations: localization. Namely, the analysis is computed for each grid-point using only local observations, without the need to build matrices that represent the entire analysis space. Localization removes long-distance correlations from $\mathbf{B}$ and allows greater flexibility in the global analysis by allowing different linear combinations of ensemble members at different spatial locations [24]. The general formulation of the LEKF by Ott goes as follows, quoting directly from [29]:

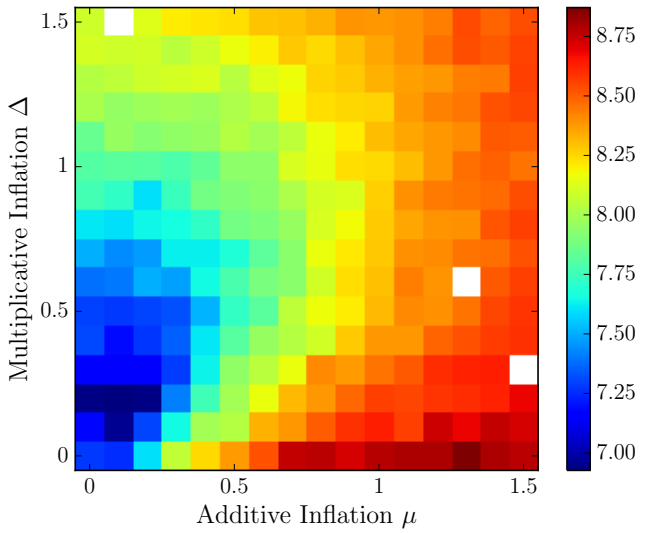

FIG. 6: The RMS error averaged over 100 model runs of length 1000 windows is reported for the ETKF for varying additive and multiplicative inflation factors Δ and μ. Each of the 100 model runs starts with a random IC, and the analysis forecast starts randomly. The window length here is 390 seconds. The filter performance RMS is computed as the RMS value of the difference between forecast and truth at the assimilation window for the latter 500 windows, allowing a spin-up of 500 windows.

1. Globally advance each ensemble member to the next analysis timestep. Steps 2–5 are performed for each grid point.
2. Create local vectors from each ensemble member.
3. Project that point's local vectors from each ensemble member into a low dimensional subspace as represented by perturbations from the mean.
4. Perform the data assimilation step to obtain a local analysis mean and covariance.
5. Generate local analysis ensemble of states.
6. Form a new global analysis ensemble from all of the local analyses.
7. Wash, rinse, and repeat.

Proposed by Hunt in 2007 with the stated objective of computational efficiency, the LETKF is named from its most similar algorithms from which it draws [19]. With the formulation of the LEKF and the ETKF given, the LETKF can be described as a synthesis of the advantages of both of these approaches. The LETKF is the method sufficiently efficient for implementation on the full OpenFOAM CFD model of 240,000 model variables, and so we present it in more detail and follow the notation of Hunt *et al.*(2007). As in the LEKF, we explicitly perform the analysis for each grid point of the model. The choice of observations to use for each grid point can be selected a priori, and tuned adaptively. Starting with a collection of background forecast vectors $\{\mathbf{x}_{b(i)} : i = 1, \ldots, k\}$, we perform steps 1 and 2 in a global variable space, then steps 3–8 for each grid point:

1. Apply H to $\mathbf{x}_{b(i)}$ to form $\mathbf{y}_{b(i)}$, average the $\mathbf{y}_b$ for $\overline{\mathbf{y}_b}$, and form $\mathbf{Y}b$.
2. Similarly form $\mathbf{X}_b$. Now for each grid point:
3. Form the local vectors.
4. Compute $\mathbf{C} = (\mathbf{Y}_b)^T\mathbf{R}^{-1}$ (perhaps by solving $\mathbf{R}\mathbf{C}^T = \mathbf{Y}_b$.
5. Compute $\tilde{\mathbf{P}}_a = ((k-1)\mathbf{I}/\rho + \mathbf{C}\mathbf{Y}_b)^{-1}$ where $\rho > 1$ is a tun-able covariance inflation factor.
6. Compute $\mathbf{W}_a = \left((k-1)\tilde{\mathbf{P}}_a\right)^{1/2}$.
7. Compute $\overline{\mathbf{w}}_a = \tilde{\mathbf{P}}_a\mathbf{C}\,(\mathbf{y}_o - \overline{\mathbf{y}}_b)$ and add it to the column of $\mathbf{W}_a$.
8. Multiply $\mathbf{X}_b$ by each $\mathbf{w}_{a(i)}$ and add $\tilde{\mathbf{x}}_b$ to get $\left\{\mathbf{x}_{a(i)} : i = 1, \ldots, k\right\}$ to complete each grid point.
9. Combine all of the local analysis into the global analysis.

We implement the LETKF on our mesh using the full 40 cells across with zone sizes of center 10, and sides 15, resulting in 1600 local variables for 100 zones. In parallel, these 100 local computations can all be carried out simultaneously over an arbitrary number of processors.

B. Adaptive covariance localization

Using the "square" sections of the loop to localize, we shift the zone to the left or right to follow the dominate flow direction at the center of that local window. In Figure 11 a schematic of localization using square, circular, and adpative location shows a situation in which adaptive localization will potentially capture more relevant information for finding the analysis state of any given cell. As we note in the caption of Figure 11, while we are motivated by localization around flow structures like Panel C, we simply shift the covariance in Panel A so that our method is most general and computationally efficient.

Denote the velocity vector of cells on a perpendicular slice of the loop at $\vec{U}$, the tangent vector to the slice $\vec{U}$ by $\vec{T}$, the zone width as $z_{\max}$ and then the localization shift α_{local} for that slice of the loop is taken to be

$$\alpha_{\text{local}} = \text{floor}\left((\vec{U}\cdot\vec{T})/\max(U) \times z_{\max}\right). \qquad (10)$$

C. Dynamic mode decomposition

We employ the "standard" algorithm of Tu to compute the Dynamic Mode Decomposition [37]. Tu's "standard" algorithm is as follows with X and Y taken as the first and last $N-1$ columns of the snapshot matrix D:

$$X = U\Sigma V \qquad \text{(Take SVD of } X.\text{)}$$
$$\tilde{A} = U^T Y V \Sigma^{-1} \qquad \text{(Build the } A \text{ matrix.)}$$
$$\tilde{A}w = \lambda w \qquad \text{(Compute eigenvectors and values.)}$$
$$\hat{\theta}w = Uw \qquad \text{(Compute corresponding modes.)}$$

Given a system state U^* we project this state onto the DMD basis by taking the real part of $U^* \cdot w = \Phi$ and use the psuedoinverse to compute the projection $(\Phi^T \cdot \Phi)^{-1} \cdot \Phi^T \cdot U^*$.

IV. RESULTS

A. Data assimilation

We confirm the performance of the DA methods described above by testing each (on the Lorenz '63 system) for increasingly long times between observations, by increasing the DA window length in Figure 5. As the time between observations increases, the nonlinearity of the Lorenz '63 system results in the failure of the EKF and difficultly for the EnKF with small ensemble size. The ETKF and EnSRF perform the best of the methods tested and we chose the ETKF for future use with the CFD model.

The results in Figure 5 rely on tuned covariance inflation, both additive and multiplicative, pre-computed for each window and DA technique. We choose optimal additive inflation μ and multiplicative inflation Δ by selecting for the lowest error in an exhaustive search through a maximum factors of 1.5 in each, an example is shown in Figure 6. We use these optimal data assimilation parameters (inflation factors) for the remainder of this work.

B. Limited observations & adaptive covariance

An initial test of prediction skill with limited observations in a twin model experiment showed that we needed 1000 spatial measurements of the temperature to predict flow reversals within 1 assimilation window. In an attempt to decrease the required observations to a experimentally realizable number, we implement a simple, adaptively localized covariance for data assimilation. Since we first saw a modest improvement in the prediction skill with full temperature observations, we hope that this improvement increases and is sufficient to get down to needing as few as 32 observations to predict reversals 1 assimilation window (of length 10 seconds) into the future.

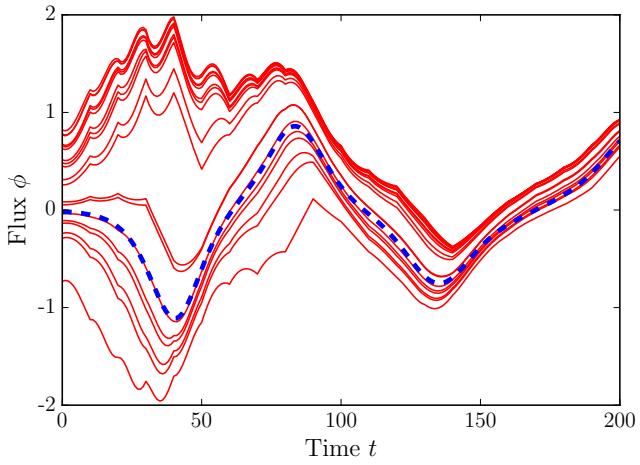

FIG. 7: Convergence of 20 ensembles using sliding windows, starting from initially random states. Here, as in most of the experiments, only temperature is observered and assimilated. Flux is computed as in Equation 4, on the left hand side of the thermosyphon, and scaled by a factor of 10^8. Assimilation takes place every 10 model seconds.

In Figure 7 we see that over an assimilation of 200 seconds, the ensemble converges on the hidden, true state. To test the performace of flow reversal prediction, we take the average of the ensemble flow direction (the average of each value of ϕ) as the predicted flow direction, and count how often we predict reversals both when they do and do not occur. Varying both the number of model variables and the strength of convariance shifting in Figure 8, we find that covariance shifting improves flow reversal prediction skill even when overservations are decreased. With full observations (spacing of 1), we obtain a the best predictions with a covariance shift of 2. For 1/2 and 1/5 observations [a spacing of 2 (5) to obvserve every other (fifth) variable], we again have the best predictions with a shift of 2. And for a spacing of 10, observing every 10th variable, we achieve greater prediction skill with a covariance shift of 10.

Computing the average flow direction inside a localized covariance zone is straighforward, and computationally easy since the velocity is immediately available, making incorporation of this scheme into any data assimilation method easy. Since observations are also sparse in large weather models, we expect that using an adaptive local covariance scheme could lead to improved prediction skill.

C. Dynamic mode decomposition

To incorporate limited observations into a high-dimensional CFD simulation, we combine ideas from both CFD literature and data assimilation to make predictions. We proceed with Tu's algorithm using snap-

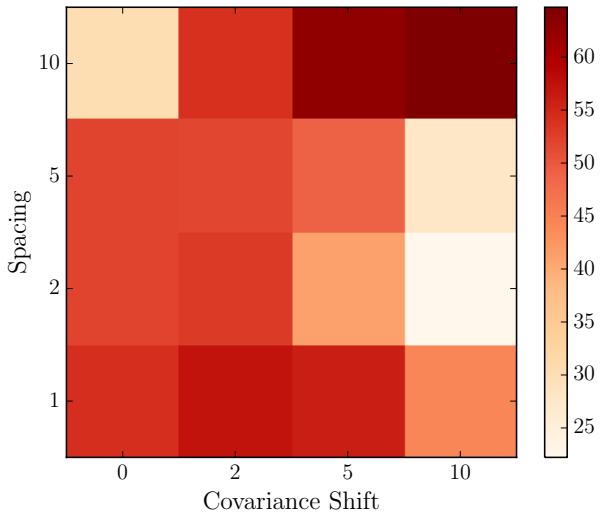

FIG. 8: Prediction skill as fraction of reversals that we correctly predicted across different numbers of observations and sliding windows of localized covariance.

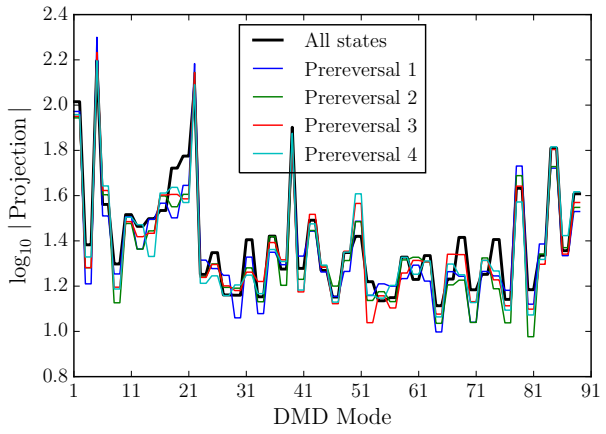

FIG. 9: The $\log_{10}$ average projection onto each DMD mode for different sets of model states. DMD constructed as snapshots every 10 seconds for the first 900 seconds of model time, and model states from the first 2000 seconds are all projected onto the DMD modes. All states average shown in black, and the average of the subset of states that occur 1 second, 3 seconds, 5 seconds, and 7 seconds before a reversal are shown in other colors. The symmetry of the loop generates modes that often come in pairs.

shots every 10 seconds for the first 900 seconds of model time. A full picture of this time series can be found Appendix C, Figure S5.

In this reduced space, we extract the modes that correspond to the instability leading to flow reversals. For modes 21 and 79, we directly observe in Figure 9 that the average projection from states just 1 second before reversal is the most different from the average stat projection, and the further away from the reversal, the more similiar the states become to the average. In Figure 10 we see that the dominant dynamics from mode 2 plotted with those of mode 79 are able to strongly separate reversal.

V. CONCLUDING REMARKS

The first output of our work is a general data assimilation framework for MATLAB and Julia. By utilizing an object-oriented (OO) design, the model and data assimilation algorithm code are separate and can be changed independently. The principal advantage of this approach is the ease of incorporation of new models and DA techniques (code available at `https://github.com/andyreagan/julia-openfoam`).

We first present the results pertaining to the accuracy of forecasts for synthetic data (twin model experiments). There are many possible experiments given the choice of assimilation window, data assimilation algorithm, localization scheme, model resolution, observational density, observed variables, and observation quality. We focused on considering the effect of observations and observational locations on the resulting forecast skill, and we find that there is a threshold for the required number of observations to make useful predictions. In general, we see that increasing observational density leads to improved forecast accuracy. With too few observations, the data assimilation is unable to recover the underlying dynamics. Using adaptively localized covariance holds promise for data assimilation with data-scarce models, to overcome the lack of data.

The ability of DMD to recover the lower dimensional dynamics is expected but with 120,000 variables is nonetheless an accomplishment. When modeling systems for which there are unknown but useful dimension reductions, as demonstrated here, DMD can be a useful tool.

The numerical coupling of CFD to experiment by DA should be generally useful to improve the skill of CFD predictions in even data-poor experiments, which can provide better knowledge of unobservable quantities of interest in fluid flow.

Acknowledgments

This work was made possible by funding from the Vermont Space Grant Consortium, NASA EPSCoR, NSF-DMS Grant No. 0940271, the Mathematics & Climate Research Network and the Vermont Advanced Computing Center.

[1] Anderson, M. J. (2001). A new method for non-parametric multivariate analysis of variance. *Austral Ecology 26*(1), 32–46.

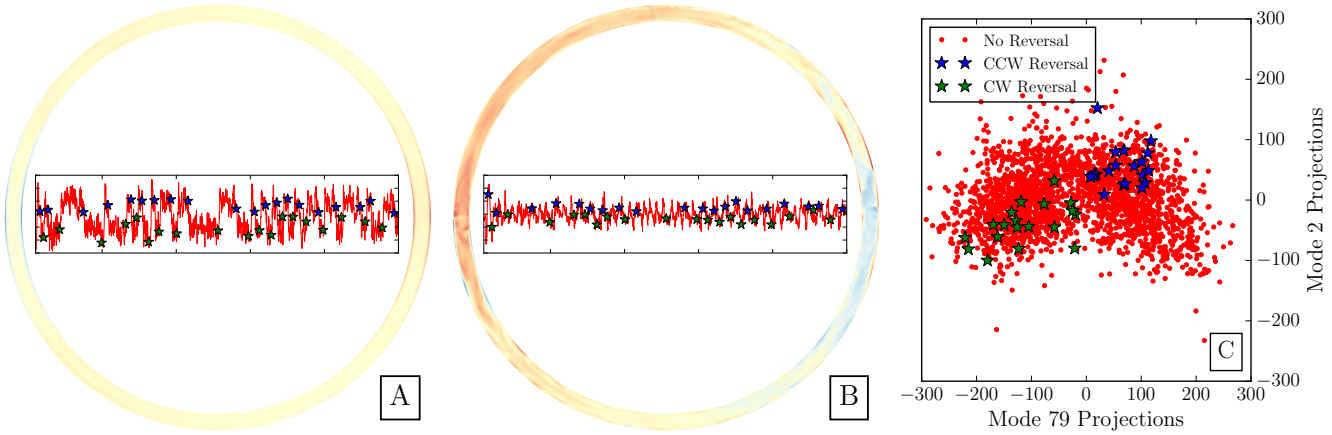

FIG. 10: Panel A: The temperature profile of the thermosyphon of Mode 2, with inset of the projection of time series states onto Mode 2 (the projection coefficient). The color scale on the thermosyphon scales from red to blue over the values 1 to 0 in the DMD mode. The inset figure is the projection coefficient from time 100 to time 5000, with the projection range being shown from -300 to 300 (as in Panel C) and the starred reversals labeled as in Panel C. Panel B: Likewise, the temperature profile of the thermosyphon of Mode 79, with inset of the projection of time series states onto Mode 79 (the projection coefficient). The color scale and inset figure axes are the same as Panel A. Panel C: A butterfly-shaped phase plane shows the value of the projection onto modes 2 and 79 for each time in the first 2000 timesteps of our ground truth model run. In blue and green stars the states that occur directly before a flow are highlighted, and are isolated into separate quadrants of phase space.

[2] Asur, S. and B. A. Huberman (2010). Predicting the future with social media. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, Volume 1, pp. 492–499. IEEE.

[3] Bauer, P., A. Thorpe, and G. Brunet (2015). The quiet revolution of numerical weather prediction. *Nature 525*(7567), 47–55.

[4] Bishop, C. H., B. J. Etherton, and S. J. Majumdar (2001). Adaptive sampling with the ensemble transform kalman filter. part i: Theoretical aspects. *Monthly weather review 129*(3), 420–436.

[5] Burgers, G., P. Jan van Leeuwen, and G. Evensen (1998). Analysis scheme in the ensemble kalman filter. *Monthly weather review 126*(6), 1719–1724.

[6] Burroughs, E., E. Coutsias, and L. Romero (2005). A reduced-order partial differential equation model for the flow in a thermosyphon. *Journal of Fluid Mechanics 543*, 203–238.

[7] Creveling, H., D. PAZ, J. Baladi, and R. Schoenhals (1975). Stability characteristics of a single-phase free convection loop. *Journal of Fluid Mechanics 67*(part 1), 65–84.

[8] Danforth, C. M., E. Kalnay, and T. Miyoshi (2007). Estimating and correcting global weather model error. *Monthly weather review 135*(2), 281–299.

[9] Desrayaud, G., A. Fichera, and M. Marcoux (2006). Numerical investigation of natural circulation in a 2d-annular closed-loop thermosyphon. *International journal of heat and fluid flow 27*(1), 154–166.

[10] Ehrhard, P. and U. Müller (1990). Dynamical behaviour of natural convection in a single-phase loop. *Journal of Fluid mechanics 217*, 487–518.

[11] Evensen, G. (2003). The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean dynamics 53*(4), 343–367.

[12] Ferziger, J. H. and M. Perić (1996). *Computational methods for fluid dynamics*, Volume 3. Springer Berlin.

[13] Ginsberg, J., M. H. Mohebbi, R. S. Patel, L. Brammer, M. S. Smolinski, and L. Brilliant (2008). Detecting influenza epidemics using search engine query data. *Nature 457*(7232), 1012–1014.

[14] Glenn, D. (2013, may). Characterizing weather in a thermosyphon: an atmosphere that hangs on a wall. Undergraduate Honors Thesis, University of Vermont.

[15] Gorman, M. and P. Widmann (1984). Chaotic flow regimes in a convection loop. *Phys. Rev. Lett.* (52), 2241–2244.

[16] Gorman, M., P. Widmann, and K. Robbins (1986). Nonlinear dynamics of a convection loop: a quantitative comparison of experiment with theory. *Physica D* (19), 255–267.

[17] Harris, K. D., E. H. Ridouane, D. L. Hitt, and C. M. Danforth (2011). Predicting flow reversals in chaotic natural convection using data assimilation. *arXiv preprint arXiv:1108.5685*.

[18] Hsiang, S. M., M. Burke, and E. Miguel (2013). Quantifying the influence of climate on human conflict. *Science 341*(6151).

[19] Hunt, B. R., E. J. Kostelich, and I. Szunyogh (2007). Efficient data assimilation for spatiotemporal chaos: A local ensemble transform kalman filter. *Physica D: Nonlinear Phenomena 230*(1), 112–126.

[20] Issa, R. I. (1986). Solution of the implicitly discretised fluid flow equations by operator-splitting. *Journal of Computational physics 62*(1), 40–65.

[21] Jasak, H., A. Jemcov, and Z. Tukovic (2007). Openfoam: A c++ library for complex physics simulations. In *International Workshop on Coupled Methods in Numerical Dynamics, IUC, Dubrovnik, Croatia*, pp. 1–20.

[22] Jiang, Y. and M. Shoji (2003). Spatial and temporal stabilities of flow in a natural circulation loop: influences of thermal boundary condition. *J Heat Trans.* (125), 612–623.

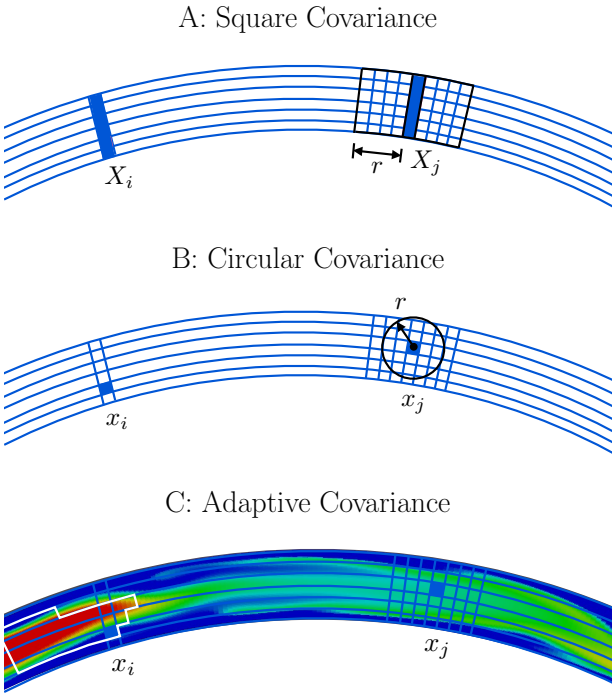

FIG. 11: Schematic of the adaptive covariance localization. In Panel A we see a zonal (square) covariance that is most similar to the covariance used for both control sliding covariance. Panel B shows a localized covariance using a "local radius", and Panel C shows an idealized, fully adaptive covariance. While we are motivated by localization around flow structures like Panel C, we simply shift the covariance in Panel A so that our method is most general and computationally efficient.

[23] Kalnay, E. (2003). *Atmospheric modeling, data assimilation, and predictability*. Cambridge university press.

[24] Kalnay, E., H. Li, T. Miyoshi, S.-C. YANG, and J. BALLABRERA-POY (2007). 4-d-var or ensemble kalman filter? *Tellus A 59*(5), 758–773.

[25] Keller, J. B. (1966). Periodic oscillations in a model of thermal convection. *J. Fluid Mech 26*(3), 599–606.

[26] Li, H., E. Kalnay, T. Miyoshi, and C. M. Danforth (2009). Accounting for model errors in ensemble data assimilation. *Monthly Weather Review 137*(10), 3407–3419.

[27] Lorenc, A. C. (1986). Analysis methods for numerical weather prediction. *Quarterly Journal of the Royal Meteorological Society 112*(474), 1177–1194.

[28] Louisos, W. F., D. L. Hitt, and C. M. Danforth (2013). Chaotic flow in a 2d natural convection loop with heat flux boundaries. *International Journal of Heat and Mass Transfer 61*, 565–576.

[29] Ott, E., B. R. Hunt, I. Szunyogh, A. V. Zimin, E. J. Kostelich, M. Corazza, E. Kalnay, D. Patil, and J. A. Yorke (2004). A local ensemble kalman filter for atmospheric data assimilation. *Tellus A 56*(5), 415–428.

[30] Patankar, S. V. and D. B. Spalding (1972). A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International Journal of Heat and Mass Transfer 15*(10), 1787–1806.

[31] Rall, L. (1981). *Automatic Differentiation: Techniques and Applications (Lecture Notes in Computer Science)*. Springer.

[32] Reagan, A. (2013). Predicting flow reversals in a computational fluid dynamics simulated thermosyphon using data assimilation. *arXiv:1312.2142*.

[33] Ridouane, E. H., C. M. Danforth, and D. L. Hitt (2010). A 2-d numerical study of chaotic flow in a natural convection loop. *International Journal of Heat and Mass Transfer 53*(1), 76–84.

[34] Savely, R., B. Cockrell, , and S. Pines (1972). Apollo experience report - onboard navigational and alignment software. *Technical Report*.

[35] Sornette, D. and W.-X. Zhou (2006). Predictability of large future changes in major financial indices. *International Journal of Forecasting 22*(1), 153–168.

[36] Tippett, M. K., J. L. Anderson, C. H. Bishop, T. M. Hamill, and J. S. Whitaker (2003). Ensemble square root filters*. *Monthly Weather Review 131*(7), 1485–1490.

[37] Tu, J. H., C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz (2013). On dynamic mode decomposition: theory and applications. *arXiv preprint arXiv:1312.0041*.

[38] Welander, P. (1995). On the oscillatory instability of a differentially heated fluid loop. *International Geophysics Series 59*.

[39] Yang, S.-C., D. Baker, H. Li, K. Cordes, M. Huff, G. Nagpal, E. Okereke, J. Villafane, E. Kalnay, and G. S. Duane (2006). Data assimilation as synchronization of truth and model: Experiments with the three-variable lorenz system*. *Journal of the atmospheric sciences 63*(9), 2340–2354.

[40] Yuen, P. and H. Bau (1999). Optimal and adaptive control of chaotic convection. *Phys. Fluids* (11), 1435–1448.

Appendix A: Computational Details and Explicit Equations Used

In this section, we first present the governing equations for the flow in our thermal convection loop experiment. A spatial and temporal discretization of the governing equations is then necessary so that they may be solved numerically. After discretization, we must specify the boundary conditions. With the mesh and boundary conditions in place, we can then simulate the flow with a computational fluid dynamics solver.

We now discuss the equations, mesh, boundary conditions, and solver in more detail. With these considerations, we present our simulations of the thermosyphon. For a complete derivation of the equations used, see [32].

1. Governing Equations

We consider the incompressible Navier-Stokes equations with the Boussinesq approximation to model the flow of water inside a thermal convection loop. Here we present the main equations that are solved numerically, noting the assumptions that are necessary in their derivation. In standard notation, for u, v, w the velocity in the x, y, z direction, respectively, the continuity equation for an incompressible fluid is

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0. \tag{A1}$$

The momentum equations, presented compactly in tensor notation with bars representing averaged quantities, are given by

$$\rho_{\text{ref}} \left(\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial}{\partial x_j} \left(\bar{u}_j \bar{u}_i \right) \right) = -\frac{\partial \bar{p}}{\partial x_i} + \mu \frac{\partial \bar{u}_i}{\partial x_j^2} + \bar{\rho} g_i \tag{A2}$$

for ρ_{ref} the reference density, ρ the density from the Boussinesq approximation, p the pressure, μ the viscocity, and g_i gravity in the i-direction. Note that $g_i = 0$ for $i \in \{x, y\}$ since gravity is assumed to be the z direction. The energy equation is given by

$$\frac{\partial}{\partial t} \left(\rho \overline{e} \right) + \frac{\partial}{\partial x_j} \left(\rho \overline{e} \overline{u}_j \right) = -\frac{\partial q_k^*}{\partial x_k} - \frac{\partial \overline{q}_k}{\partial x_k} \tag{A3}$$

for e the total internal energy and q the flux (where $q = \overline{q} + q^*$ is the averaging notation).

2. Implementation

The PISO (Pressure-Implicit with Splitting of Operators) algorithm derives from the work of [20], and is complementary to the SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) [30] iterative method. The main difference of the PISO and SIMPLE algorithms is that in the PISO, no under-relaxation is applied and the momentum corrector step is performed more than once [12]. They sum up the algorithm in nine steps:

- Set the boundary conditions.
- Solve the discretized momentum equation to compute an intermediate velocity field.
- Compute the mass fluxes at the cell faces.
- Solve the pressure equation.
- Correct the mass fluxes at the cell faces.
- Correct the velocity with respect to the new pressure field.
- Update the boundary conditions.
- Repeat from step #3 for the prescribed number of times.
- Repeat (with increased time step).

The solver itself has 647 dependencies, of which I present only a fraction. The main code is straight forward, relying on include statements to load the libraries and equations to be solved.

```
#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "RASModel.H" // AJR edited 2013-10-14
#include "radiationModel.H"
#include "fvIOoptionList.H"
#include "pimpleControl.H"
```

The main function is then

```
int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "readGravitationalAcceleration.H"
    #include "createFields.H"
    #include "createIncompressibleRadiationModel.H"
    #include "createFvOptions.H"
    #include "initContinuityErrs.H"
    #include "readTimeControls.H"
    #include "CourantNo.H"
    #include "setInitialDeltaT.H"
    pimpleControl pimple(mesh);
```

We then enter the main loop. This is computed for each time step, prescribed before the solver is applied. Note that the capacity is available for adaptive time steps, choosing to keep the Courant number below some threshold, but I do not use this. For the distributed ensemble of model runs, it is important that each model complete in nearly the same time, so that the analysis is not waiting on one model and therefore under-utilizing the available resources.

```
    while (runTime.loop())
    {
        #include "readTimeControls.H"
        #include "CourantNo.H"
        #include "setDeltaT.H"
        while (pimple.loop())
        {
            #include "UEqn.H"
            #include "TEqn.H"
            while (pimple.correct())
            {
                #include "pEqn.H"
            }
        }
        if (pimple.turbCorr())
        {
            turbulence->correct();
        }
```

Opening up the equation for U we see that Equation

```
    // Solve the momentum equation
    fvVectorMatrix UEqn
    (
        fvm::ddt(U)
      + fvm::div(phi, U)
      + turbulence->divDevReff(U)
     ==
        fvOptions(U)
    );
    UEqn.relax();
    fvOptions.constrain(UEqn);
    if (pimple.momentumPredictor())
    {
        solve
        (
            UEqn
         ==
            fvc::reconstruct
            (
                (
                  - ghf*fvc::snGrad(rhok)
                  - fvc::snGrad(p_rgh)
                )*mesh.magSf()
            )
        );
```

```
        fvOptions.correct(U);
    }
```

Solving for T is

```
{
    alphat = turbulence->nut()/Prt;
    alphat.correctBoundaryConditions();
    volScalarField alphaEff("alphaEff", turbulence->nu()/Pr + alphat);
    fvScalarMatrix TEqn
    (
        fvm::ddt(T)
      + fvm::div(phi, T)
      - fvm::laplacian(alphaEff, T)
     ==
        radiation->ST(rhoCpRef, T)
      + fvOptions(T)
    );
    TEqn.relax();
    fvOptions.constrain(TEqn);
    TEqn.solve();
    radiation->correct();
    fvOptions.correct(T);
    rhok = 1.0 - beta*(T - TRef); // Boussinesq approximation
}
```

Finally, we solve for the pressure p in "pEqn.H":

```
{
    volScalarField rAU("rAU", 1.0/UEqn.A());
    surfaceScalarField rAUf("Dp", fvc::interpolate(rAU));
    volVectorField HbyA("HbyA", U);
    HbyA = rAU*UEqn.H();
    surfaceScalarField phig(-rAUf*ghf*fvc::snGrad(rhok)*mesh.magSf());
    surfaceScalarField phiHbyA
    (
        "phiHbyA",
        (fvc::interpolate(HbyA) & mesh.Sf())
      + fvc::ddtPhiCorr(rAU, U, phi)
      + phig
    );
    while (pimple.correctNonOrthogonal())
    {
        fvScalarMatrix p_rghEqn
        (
            fvm::laplacian(rAUf, p_rgh) == fvc::div(phiHbyA)
        );
        p_rghEqn.setReference(pRefCell, getRefCellValue(p_rgh, pRefCell));
        p_rghEqn.solve(mesh.solver(p_rgh.select(pimple.finalInnerIter())));
        if (pimple.finalNonOrthogonalIter())
        {
            // Calculate the conservative fluxes
            phi = phiHbyA - p_rghEqn.flux();
            // Explicitly relax pressure for momentum corrector
            p_rgh.relax();
            // Correct the momentum source with the pressure gradient flux
            // calculated from the relaxed pressure
            U = HbyA + rAU*fvc::reconstruct((phig - p_rghEqn.flux())/rAUf);
            U.correctBoundaryConditions();
        }
    }
    #include "continuityErrs.H"
    p = p_rgh + rhok*gh;
    if (p_rgh.needReference())
    {
        p += dimensionedScalar
        (
            "p",
            p.dimensions(),
            pRefValue - getRefCellValue(p, pRefCell)
        );
        p_rgh = p - rhok*gh;
    }
}
```

The final operation being the conversion of pressure to hydrostatic pressure,

$$p_{\mathrm{rgh}} = p - \rho_k g_h.$$

This "pEqn.H" is then re-run until convergence is achieved, and the PISO loop begins again.

Appendix B: The Ehrhard and Müller Equations

Following the derivation by Harris [17], itself a representation of the derivation of Gorman [16] and namesakes Ehrhard and Müller [10], we derive the equations governing a closed loop thermosyphon.

Similar to the derivation of the governing equations of computational fluid dynamics, we start with a small but finite volume inside the loop. Here, however, the volume is described by $\pi r^2 R \mathrm{d}\phi$ for r the interior loop size (such that πr^2 is the area of a slice) and $R\mathrm{d}\phi$ the arc length (width) of the slice. Newton's second law states that momentum is conserved, such that the sum of the forces acting upon our finite volume is equal to the change in momentum of this volume. Therefore we have the basic starting point for forces $\sum F$ and velocity u as

$$\sum F = \rho \pi r^2 R \mathrm{d}\phi \frac{\mathrm{d}u}{\mathrm{d}t}. \tag{B1}$$

The sum of the forces is $\sum F = F_{\{p,f,g\}}$ for net pressure, fluid shear, and gravity, respectively. We write these as

$$F_p = -\pi r^2 \mathrm{d}\phi \frac{\partial p}{\partial \phi} \tag{B2}$$

$$F_w = -\rho \pi r^2 \mathrm{d}\phi f_w \tag{B3}$$

$$F_g = -\rho \pi r^2 \mathrm{d}\phi g \sin(\phi) \tag{B4}$$

where $\partial p/\partial \phi$ is the pressure gradient, f_w is the wall friction force, and $g\sin(\phi)$ is the vertical component of gravity acting on the volume.

We now introduce the Boussinesq approximation which states that both variations in fluid density are linear in temperature T and density variation is insignificant except when multiplied by gravity. The consideration manifests as

$$\rho = \rho(T) \simeq \rho_{\mathrm{ref}}(1 - \beta(T - T_{\mathrm{ref}})$$

where ρ_0 is the reference density and T_{ref} is the reference temperature, and β is the thermal expansion coefficient. The second consideration of the Boussinesq approximation allows us to replace ρ with this ρ_{ref} in all terms except for F_g. We now write momentum equation as

$$-\pi r^2 \mathrm{d}\phi \frac{\partial p}{\partial \phi} - \rho_{\mathrm{ref}} \phi r^2 R \mathrm{d}\phi f_w - \rho_{\mathrm{ref}}(1 - \rho(T - T_{\mathrm{ref}}))\pi r^2 R \mathrm{d}\phi g \sin(\phi) = \rho_{\mathrm{ref}} \pi r^2 R \mathrm{d}\phi \frac{\mathrm{d}u}{\mathrm{d}t}. \tag{B5}$$

Canceling the common πr^2, dividing by R, and pulling out $\mathrm{d}\phi$ on the LHS we have

$$-\mathrm{d}\phi \left(\frac{\partial p}{\partial \phi} \frac{1}{R} - \rho_{\mathrm{ref}} f_w - \rho_{\mathrm{ref}}(1 - \rho(T - T_{\mathrm{ref}})) g \sin(\phi) \right) = \rho_{\mathrm{ref}} \mathrm{d}\phi \frac{\mathrm{d}u}{\mathrm{d}t}. \tag{B6}$$

We integrate this equation over ϕ to eliminate many of the terms, specifically we have

$$\int_0^{2\pi} -\mathrm{d}\phi \frac{\partial p}{\partial \phi} \frac{1}{R} \to 0$$

$$\int_0^{2\pi} -\mathrm{d}\phi \rho_{\mathrm{ref}} g \sin(\phi) \to 0$$

$$\int_0^{2\pi} -\mathrm{d}\phi \rho_{\mathrm{ref}} \beta T_{\mathrm{ref}} g \sin(\phi) \to 0.$$

Since u (and hence $\frac{\mathrm{d}u}{\mathrm{d}\phi}$) and f_w do not depend on ϕ, we can pull these outside an integral over ϕ and therefore the momentum equation is now

$$2\pi f_w \rho_0 + \int_0^{2\pi} \mathrm{d}\phi \rho_{\mathrm{ref}} \beta T g \sin(\phi) = 2\pi \frac{\mathrm{d}u}{\mathrm{d}\phi} \rho_{\mathrm{ref}}.$$

Diving out 2π and pull constants out of the integral we have our final form of the momentum equation

$$f_w \rho_{\text{ref}} + \frac{\rho_{\text{ref}} \beta g}{2\pi} \int_0^{2\pi} \mathrm{d}\phi T \sin(\phi) = \frac{\mathrm{d}u}{\mathrm{d}\phi} \rho_{\text{ref}}. \tag{B7}$$

Now considering the conservation of energy within the thermosyphon, the energy change within a finite volume must be balanced by transfer within the thermosyphon and to the walls. The internal energy change is given by

$$\rho_{\text{ref}} \pi r^2 R \mathrm{d}\phi \left(\frac{\partial T}{\partial t} + \frac{u}{R} \frac{\partial T}{\partial \phi} \right) \tag{B8}$$

which must equal the energy transfer through the wall, which is, for T_w the wall temperature:

$$\dot{q} = -\pi r^2 R \mathrm{d}\phi h_w (T - T_w). \tag{B9}$$

Combining Equations B8 and B9 (and canceling terms) we have the energy equation:

$$\left(\frac{\partial T}{\partial t} + \frac{u}{R} \frac{\partial T}{\partial \phi} \right) = \frac{-h_w}{\rho_{\text{ref}} c_p} \left(T - T_w \right). \tag{B10}$$

The f_w which we have yet to define and h_w are fluid-wall coefficients and can be described by [10]:

$$\begin{aligned} h_w &= h_{w_0} \left(1 + K h(|x_1|) \right) \\ f_w &= \frac{1}{2} \rho_{\text{ref}} f_{w_0} u. \end{aligned}$$

We have introduced an additional function h to describe the behavior of the dimensionless velocity $x_1 \alpha u$. This function is defined piece-wise as

$$h(x) = \begin{cases} x^{1/3} & \text{when } x \geq 1 \\ p(x) & \text{when } x < 1 \end{cases} \tag{B11}$$

where $p(x)$ can be defined as $p(x) = \left(44x^2 - 55x^3 + 20x^4 \right)/9$ such that p is analytic at 0 [17].

Taking the lowest modes of a Fourier expansion for T for an approximate solution, we consider:

$$T(\phi, t) = C_0(t) + S(t) \sin(\phi) + C(t) \cos(\phi). \tag{B12}$$

By substituting this form into Equations B7 and B10 and integrating, we obtain a system of three equations for our solution. We then follow the particular nondimensionalization choice of Harris et al such that we obtain the following ODE system, which we refer to as the Ehrhard-Müller equations:

$$\frac{\mathrm{d}x_1}{\mathrm{d}t'} = \alpha(x_2 - x_1), \tag{B13}$$

$$\frac{\mathrm{d}x_2}{\mathrm{d}t'} = \beta x_1 - x_2(1 + K h(|x_1|)) - x_1 x_3, \tag{B14}$$

$$\frac{\mathrm{d}x_3}{\mathrm{d}t'} = x_1 x_2 - x_3(1 + K h(|x_1|)). \tag{B15}$$

The nondimensionalization is given by the change of variables

$$t' = \frac{h_{w_0}}{\rho_{\text{ref}} c_p} t, \tag{B16}$$

$$x_1 = \frac{\rho_{\text{ref}} c_p}{R h_{w_0}} u, \tag{B17}$$

$$x_2 = \frac{1}{2} \frac{\rho_{\text{ref}} c_p \beta g}{R h_{w_0} f_{w_0}} \Delta T_{3-9}, \tag{B18}$$

$$x_3 = \frac{1}{2} \frac{\rho_{\text{ref}} c_p \beta g}{R h_{w_0} f_{w_0}} \left(\frac{4}{\pi} \Delta T_w - \Delta T_{6-12} \right) \tag{B19}$$

and

$$\alpha = \frac{1}{2} R c_p f_{w_0} / h_{w_0}, \tag{B20}$$

$$\gamma = \frac{2}{\pi} \frac{\rho_{\text{ref}} c_p \beta g}{R h_{w_0} f_{w_0}} \Delta T_w. \tag{B21}$$

Through careful consideration of these non-dimensional variable transformations we verify that x_1 is representative of the mean fluid velocity, x_2 of the temperature difference between the 3 and 9 o'clock positions on the thermosyphon, and x_3 the deviation from the vertical temperature profile in a conduction state [17].

Appendix C: Data Assimiliation

The TLM is the model which advances an initial perturbation $\delta\mathbf{x}_i$ at timestep i to a final perturbation $\delta\mathbf{x}_{i+1}$ at timestep $i+1$. The dynamical system we are interested in, Lorenz '63, is given as a system of ODE's:

$$\frac{d\mathbf{x}}{dt} = F(\mathbf{x}).$$

We integrate this system using a numerical scheme of our choice (in the given examples we use a second-order Runge-Kutta method), to obtain a model M discretized in time.

$$\mathbf{x}(t) = M[\mathbf{x}(t_0)].$$

Introducing a small perturbation $\mathbf{y}$, we can approximate our model M applied to $\mathbf{x}(t_0) + \mathbf{y}(t_0)$ with a Taylor series around $\mathbf{x}(t_0)$:

$$\begin{aligned} M[\mathbf{x}(t_0) + \mathbf{y}(t_0)] &= M[\mathbf{x}(t_0)] + \frac{\partial M}{\partial \mathbf{x}}\mathbf{y}(t_0) + O[\mathbf{y}(t_0)^2] \\ &\approx \mathbf{x}(t) + \frac{\partial M}{\partial \mathbf{x}}\mathbf{y}(t_0). \end{aligned}$$

We can then solve for the linear evolution of the small perturbation $\mathbf{y}(t_0)$ as

$$\frac{d\mathbf{y}}{dt} = \mathbf{J}\mathbf{y} \tag{C1}$$

where $\mathbf{J} = \partial F/\partial \mathbf{x}$ is the Jacobian of F. We can solve the above system of linear ordinary differential equations using the same numerical scheme as we did for the nonlinear model.

One problem with solving the system of equations given by Equation C1 is that the Jacobian matrix of discretized code is not necessarily identical to the discretization of the Jacobian operator for the analytic system. This is a problem because we need to have the TLM of our model M, which is the time-space discretization of the solution to $d\mathbf{x}/dt = F(\mathbf{x})$. We can apply our numerical method to the $d\mathbf{x}/dt = F(\mathbf{x})$ to obtain M explicitly, and then take the Jacobian of the result. This method is, however, prohibitively costly, since Runge-Kutta methods are implicit. It is therefore desirable to take the derivative of the numerical scheme directly, and apply this differentiated numerical scheme to the system of equations $F(\mathbf{x})$ to obtain the TLM. A schematic of this scenario is illustrated in Figure S2. To that the derivative of numerical code for implementing the EKF on models larger than 3 dimensions (i.e. global weather models written in Fortan), automatic code differentiation is used [31].

To verify our implementation of the TLM, we propagate a small error in the Lorenz '63 system and plot the difference between that error and the TLM predicted error, for each variable (Figure S1).

With a finite ensemble size, the ensemble method is only an approximation and therefore in practice it often fails to capture the full spread of error. To better capture the model variance, additive and multiplicative inflation factors are used to obtain a good estimate of the error covariance matrix (Section 2.6). The spread of ensemble members in the x_1 variable of the Lorenz model, as distance from the analysis, can be seen in Figure S3.

In computing the error covariance $\mathbf{P}_f$ from the ensemble, we wish to add up the error covariance of each forecast with respect to the mean forecast. But this would underestimate the error covariance, since the forecast we're comparing against is used in the ensemble average (to obtain the mean forecast). Therefore, to compute the error covariance matrix for each forecast, that forecast itself is excluded from the ensemble average forecast.

We can see the classic spaghetti of the ensemble with this filter implemented on Lorenz 63 in Figure S4.

We denote the forecast within an ensemble filter as the average of the individual ensemble forecasts, and an explanation for this choice is substantiated by Burgers [5]. The general EnKF which we use is most similar to that of Burgers. Many algorithms based on the EnKF have been proposed and include the Ensemble Transform Kalman Filter (ETKF) [29], Ensemble Analysis Filter (EAF) [1], Ensemble Square Root Filter (EnSRF) [36], Local Ensemble Kalman Filter (LEKF) [29], and the Local Ensemble Transform Kalman Filter (LETKF) [19]. A comprehensive overview through 2003 is provided by Evensen [11]. For further details on the most advanced methods, beyond what is provided in the body of the paper, we direct the reader the above references and the derivations provided in [32].

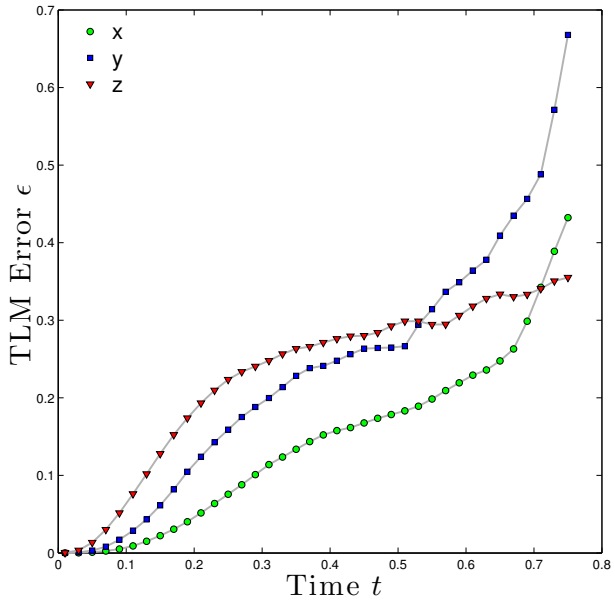

FIG. S1: The future error predicted by the TLM is compared to the error growth in Lorenz '63 system for an initial perturbation with standard deviation of 0.1, averaged over 1000 TLM integrations. The ϵ is not the error predicted by the TLM, but rather the error of the TLM in predicting the error growth.

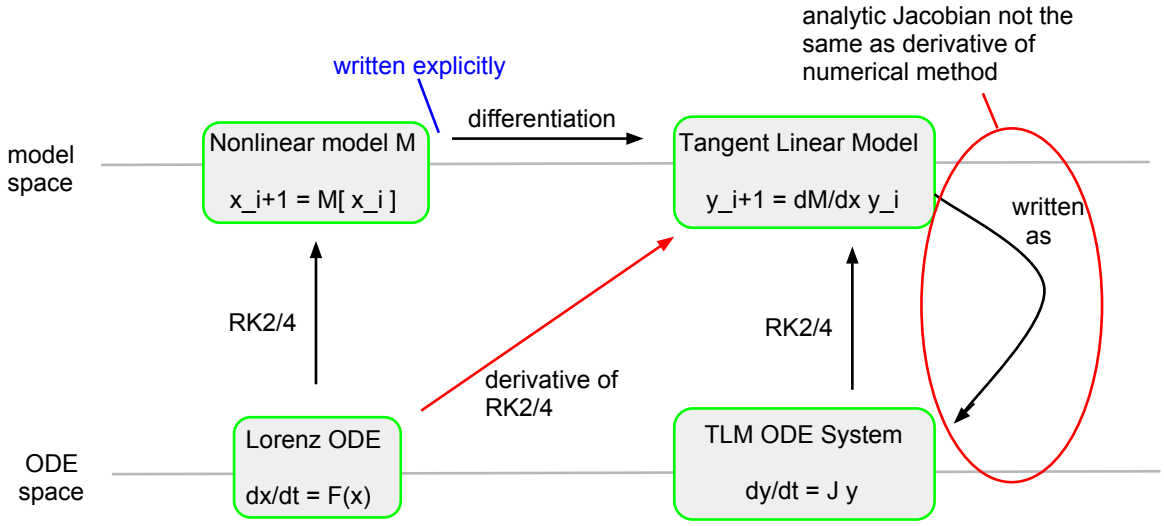

FIG. S2: An explanation of how and why the best way to obtain a TLM is with a differentiated numerical scheme.

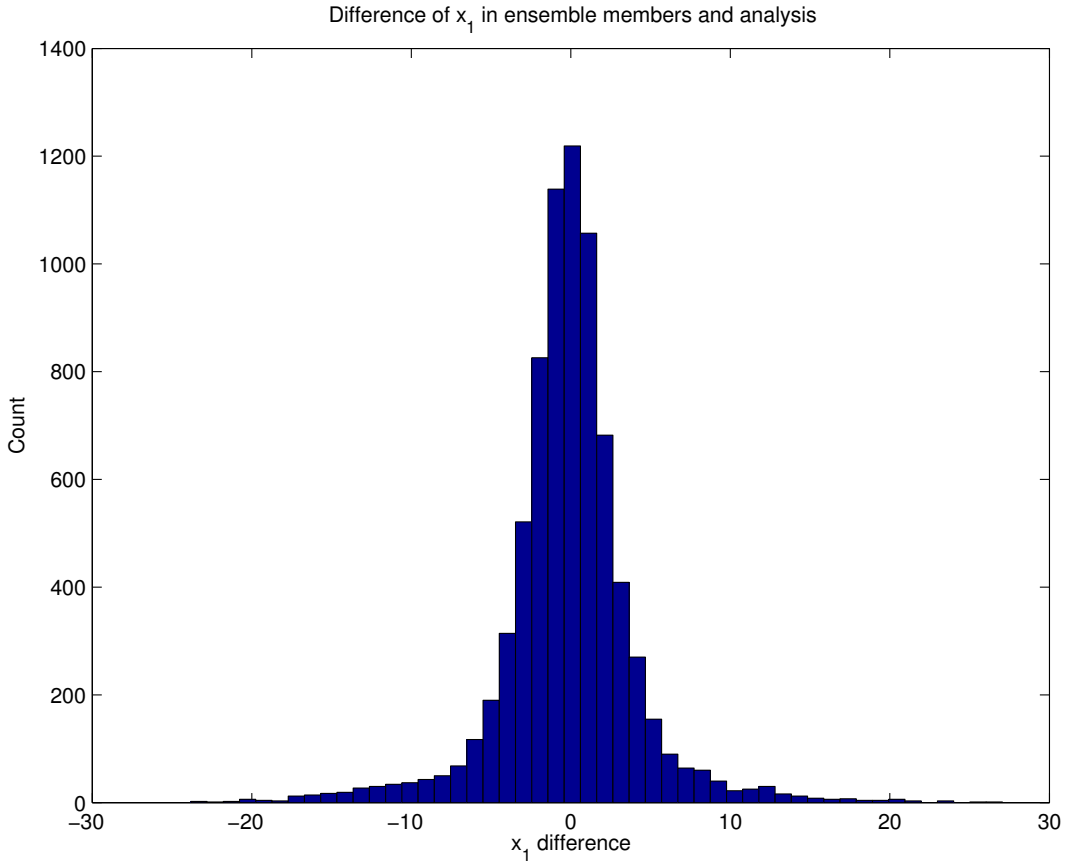

FIG. S3: The difference of ensemble forecasts from the analysis is reported for 760 assimilation windows in one model run of length 200, with 10 ensemble members and an assimilation window of length 0.261. This has the same shape of as the difference between ensemble forecasts and the mean of the forecasts (not shown). This spread of ensemble forecasts is what allows us to estimate the error covariance of the forecast model, and appears to be normally distributed.

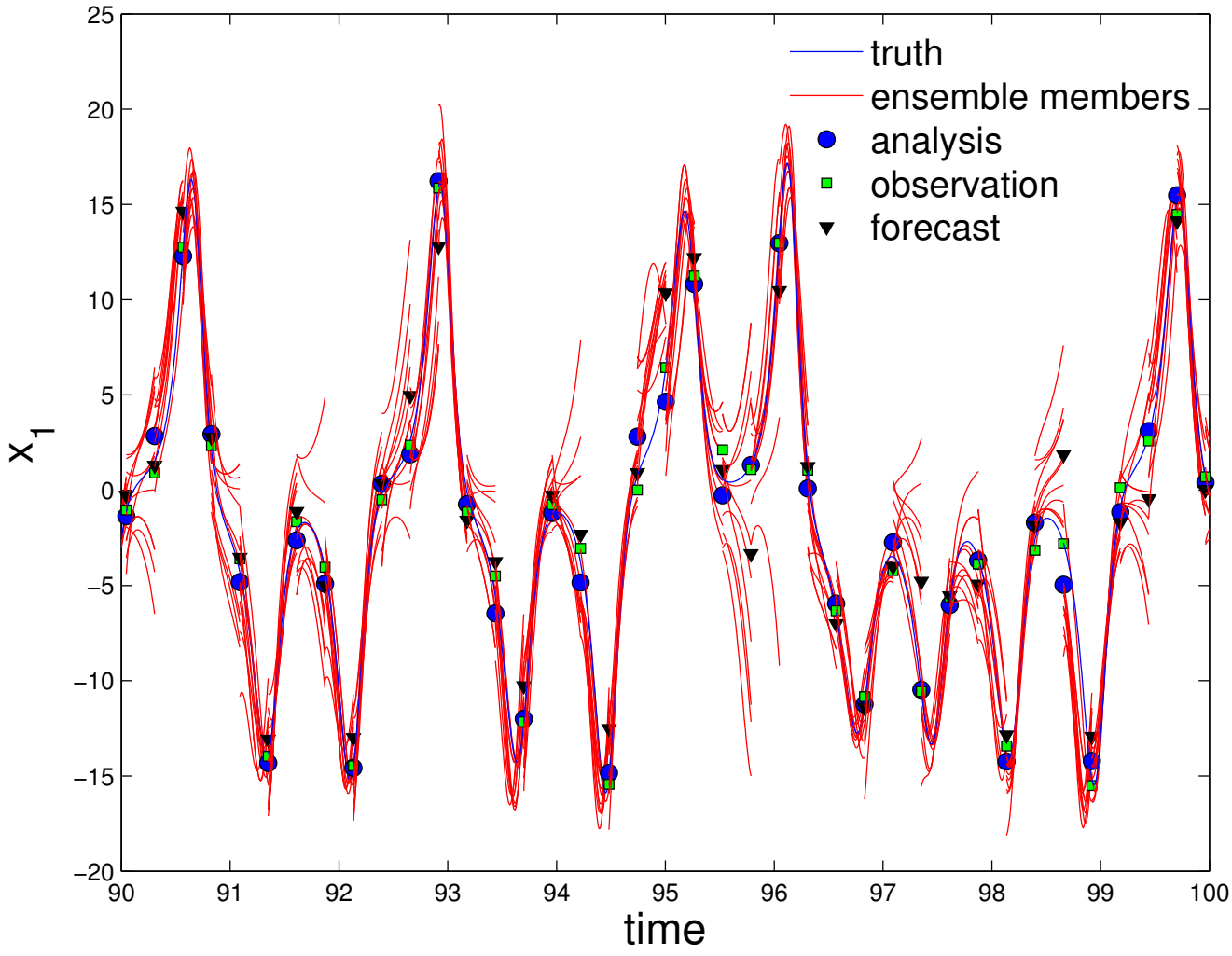

FIG. S4: A sample time-series of the ensembles used in the EnKF. In all tests, as seen here, 10 ensemble members are used. For this run, 384 assimilation cycles are performed with a window length of 0.26 model time units.

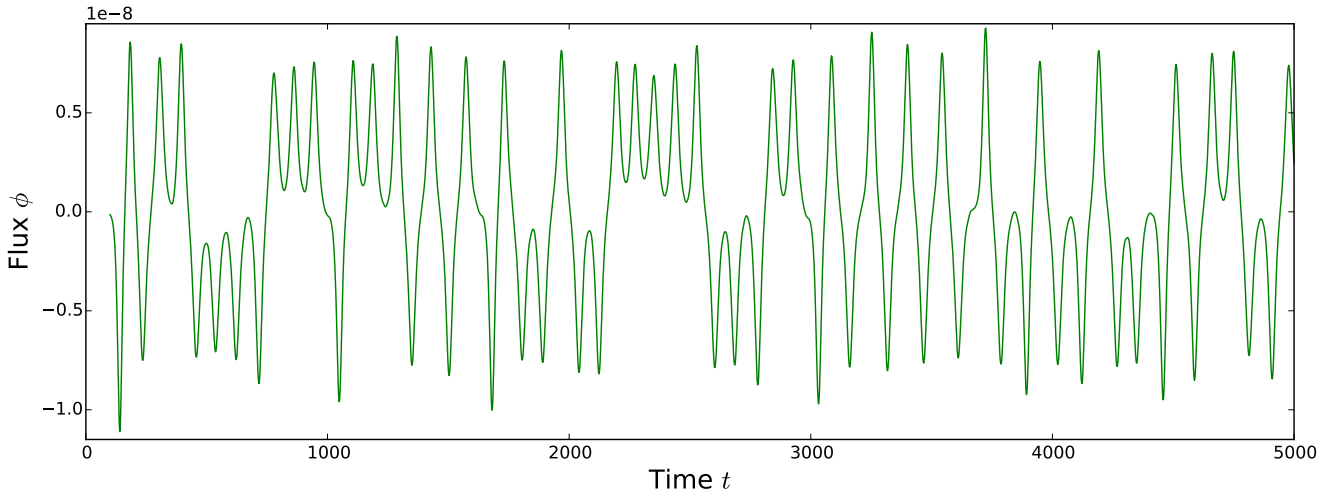

FIG. S5: Flux timeseries on which DMD is performed.

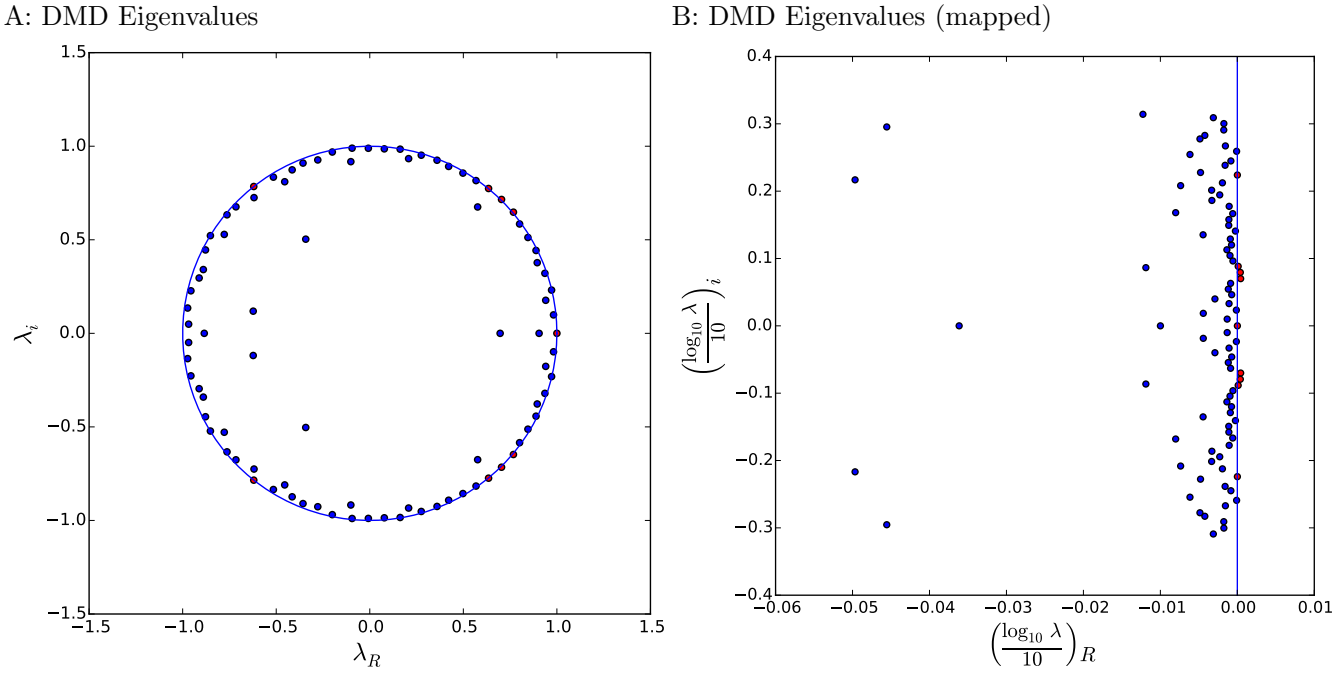

FIG. S6: Eigenvalues of DMD Modes.

Appendix D: Additional DMD Details

The general algoritm for DMD is provided in Section III(C), and here we supply more results of the DMD procedure. The timeseries from which we computed the decomposition is shownn in Figure S5.

The real and imaginary components of the DMD eigenvalues are shown in both unmapped and mapped form in Figure S6.