ABSTRACT

Title of Thesis: DATA-DRIVEN WILDFIRE PROPAGATION MODELING WITH FARSITE-ENKF

Maria Faye Theodori, M.S., 2016

Thesis Directed By: Professor Arnaud Trouvé,
Department of Fire Protection Engineering

The goal of this study is to provide a framework for future researchers to understand and use the FARSITE wildfire-forecasting model with data assimilation. Current wildfire models lack the ability to provide accurate prediction of fire front position faster than real-time. When FARSITE is coupled with a recursive ensemble filter, the data assimilation forecast method improves. The scope includes an explanation of the standalone FARSITE application, technical details on FARSITE integration with a parallel program coupler called OpenPALM, and a model demonstration of the FARSITE-Ensemble Kalman Filter software using the FireFlux I experiment by Craig Clements. The results show that the fire front forecast is improved with the proposed data-driven methodology than with the standalone FARSITE model.

DATA-DRIVEN WILDFIRE PROPAGATION MODELING WITH FARSITE-ENKF

by

Maria Faye Theodori

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2016

Advisory Committee:
Professor Arnaud Trouvé, Chair
Assistant Professor Michael Gollner
Professor and Department Chair James Milke

Acknowledgements

I would first like to thank my friend and research partner, Cong Zhang, who guided me so generously along each step of the way and helped me to grasp every important detail of this project. I truly appreciate how you went out of your way to make sure I was prepared with all the material I needed. I cannot thank you enough! I would also like to thank my defense committee, including my advisor, Professor Arnaud Trouvè, for the expertise, patience, and instruction as I learned a topic and skills that I have never been faced with before; my pseudo-co-advisor, Professor Michael Gollner, who graciously encouraged and helped me to master this project; and Professor James Milke, who for the past 5 years has given me continuous support academically, professionally, and personally. Finally, I would like to thank Mèlanie Rochoux for being the mastermind of development for us throughout the entire research process. Your skillset is invaluable and none of this would have been possible without you. Thank you for spending so much of your time to make this model come to light.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Motivation

Wildfires common in western United States, Australia, Portugal and other dry, forested parts of the world are a threat to people, property, and the environment. Understanding and predicting the movement of these destructive fires has many positive implications. Forecast of a wildfire can allow firefighters to effectively and safely suppress the flames, housing developers to understand the risks of building in certain areas, and communities to timely preplan an evacuation strategy. Modeling fires to obtain a forecast requires knowledge of the interaction between physics, fluids, chemistry, and thermodynamics. When considering the movement of a wildfire, other environmental factors must also be included such as dynamic wind properties, topography, and fuel loading. The complex relationship that these factors have, in addition to how the fire influences the atmosphere and vice versa, makes forecasting a wildfire extremely difficult. However, several wildfire models exist with the capability to provide useful insight to predicting fire propagation.

There are two types of existing wildfire models that either serve an operational purpose or a research purpose. Operational-level models, used as tools to respond to actively burning fires or for "gaming" purposes, include FARSITE (Finney, 1998) and PROMETHEUS (Tymstra et al., 2010). Research-level models

used as mathematical analogues or statistical data include WFDS (Mell, 2010), WRF-Fire (Coen, 2013), and FIRETEC (Linn and Harlow, 1998). These models rely heavily on assumptions of initial conditions, which inhibit the ability to simulate a fireline location that is entirely accurate with observed data. Advanced computer modeling methods are now being explored to improve these wildfire models and correct the fireline prediction. Data assimilation is a method that has proven successful in numerical weather prediction and is a promising technique in wildfire forecast. As previously stated, the benefits from accurately predicting wildfire propagation range from firefighter safety to community planning to atmospheric pollution reduction. It is of interest to the fire protection community, the atmospheric sciences community, and the urban planning community to improve these forecast models and increase accuracy of wildfire propagation prediction.

1.2 Overview of Data Assimilation

Geophysical modeling of weather, oceanography, atmospheric chemistry, and wildfire is comprised of complex physics given heterogeneous and uncertain sources of data. Data assimilation (DA) is a method of characterizing the system and its evolution by optimizing the various information provided to the model. The technique combines numerical models with observed measurements to provide the best possible insight into the dynamics of a future state of the given system. Estimation of the *state variables*, functions of space and time, and the *field*, the system under observation,

requires iterating a feedback loop algorithm with a forward model and an inverse model, providing a distribution or evolution of variables in three or four dimensions (Robinson et al. 2000). The process is illustrated in Figure 1.1, image by Cong Zhang, where control vector y_{t-1}^{f} holds a set of state variables (wind speed, fuel properties) to be corrected through DA in a parameter estimation strategy. The forward modeling transformation, *H*, uses observation vector y_t^o, as the target destination of the state that contains y_t^f. One run of the transformation is considered a deterministic process and produces the new set of state variables, y_t^f. Data assimilation occurs when y_t^f undergoes inversion back through *H* to produce y_{t-1}^a in the previous state, which becomes a more accurate estimate of the true state y_{t-1}^o. Then, y_{t-1}^a is mapped to the 0th state through *H* again using corrected prior state variables to produce y_t^a, an updated, more accurate estimate of the observation y_t^o.

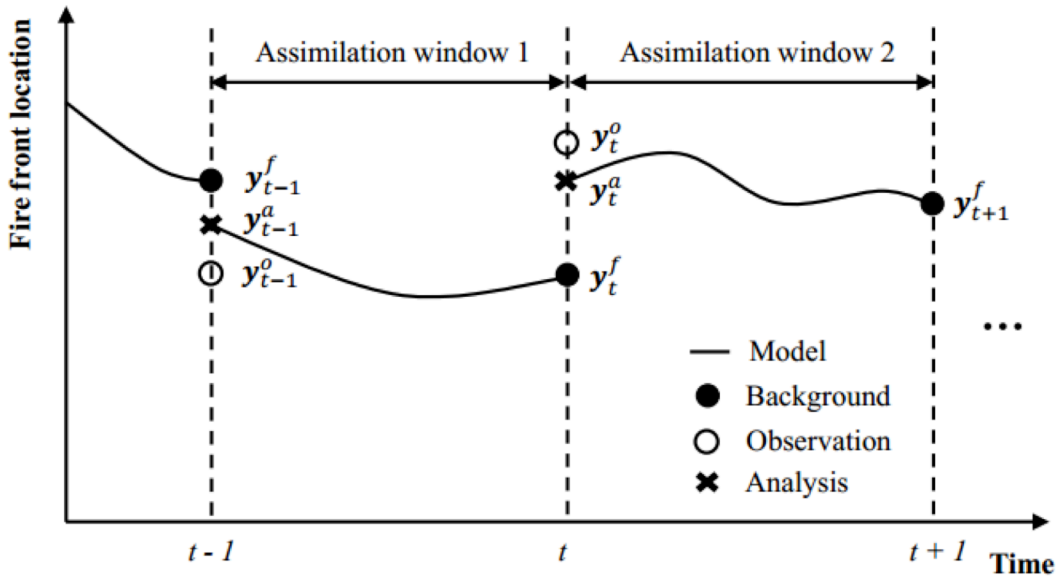

Figure 1.1 Data assimilation presented as an iterative loop that updates forward model parameters to provide an optimized state estimation. Image by Cong Zhang, 2015.

The estimates at each time step have an associated uncertainty that must be specified in terms of a background error covariance matrix. Understanding the distribution of the probability of this error is a key component to improve the DA method. The background error covariance matrix is defined for a certain number of variables, which gives the matrix its dimensions, and is assumed constant over time. The average of the errors produced by the DA model over a large number of cases accounts for the "windshield wiper" effect, a forecast consistently changing back and forth to either side of the actual value or the mean forecast (WMO, 2013). This means that the effect is averaged out too, however, the error is still present and now unquantifiable when considering a single deterministic run. One way to overcome this effect is by using an ensemble DA approach, in which multiple deterministic outputs are produced at once. If an ensemble model has enough members, or deterministic inputs, it can reduce the uncertainty of error.

An ensemble DA simulation can be thought of in the context of fire as a fan of data points that create a fireline spreading outward from an initial ignition point. Incorporating probabilistic physics into the observation forecast at each time step creates these data points. As the number of ensembles is increased through a given zone, the confidence in forecast accuracy increases, whereas a decreasing percentage of ensemble members lessen the probability of fire spread in that zone. The computational cost of running a 10 member ensemble model is comparable to the cost of running the deterministic model at twice the resolution, but the statistical proficiency is greatly improved with the ensemble run (Gall et al., 2013). Inevitably,

there are a number of difficulties with DA due to nonlinearities, multiple scales, unknown error statistics, and non-reproducibility of the unique forecasts.

1.2.1 Ensemble Kalman Filter

There exists a variety of data assimilation schemes featuring different feedback loop algorithms and mathematical inverse models. The distribution of information from one time step at one location to the next requires knowledge about the connection between the variables. The Ensemble Kalman Filter (EnKF) aids in this aspect by carrying out the calculations in the space spanned by the ensemble members in a localized and sequential way. It is a mathematical technique used in geophysical data assimilation methods to incorporate a Gaussian distribution of probability for dynamic events that are unknown. This recursive ensemble filter is used to modify the state as the system runs. The model advances in time with newly provided data from the ensemble at each time step. This produces an optimal combination of both measurements and model estimation (Rochoux, 2014). Figure 1.2, image by Mèlanie Rochoux, displays this process in a flowchart where initial conditions and boundary conditions are inputs to the forward model, the model outputs are adjusted based on comparison to observation and state estimation, and then the updated parameters are fed back into the forward model. The error that propagates from the quantified difference of real data to estimate data can be explicitly known based on the predetermined error covariance matrix.

This simulation is considered a Monte Carlo approximation, in which the user understands that the uncertainty in forecasting is based on an ensemble of model trajectories and uses repeated random samplings to obtain numerical results. Conducting parameter estimation with this approach allows for flow-dependent error covariances and accounts for some of the nonlinearities in the spatial and temporal integration of the transformation operator *H*. The EnKF algorithm approximates the mean and the covariance of the background state by the mean and covariance of the ensemble, using an assumption that all the probability density functions (PDFs) are Gaussian (Durand, 2015).

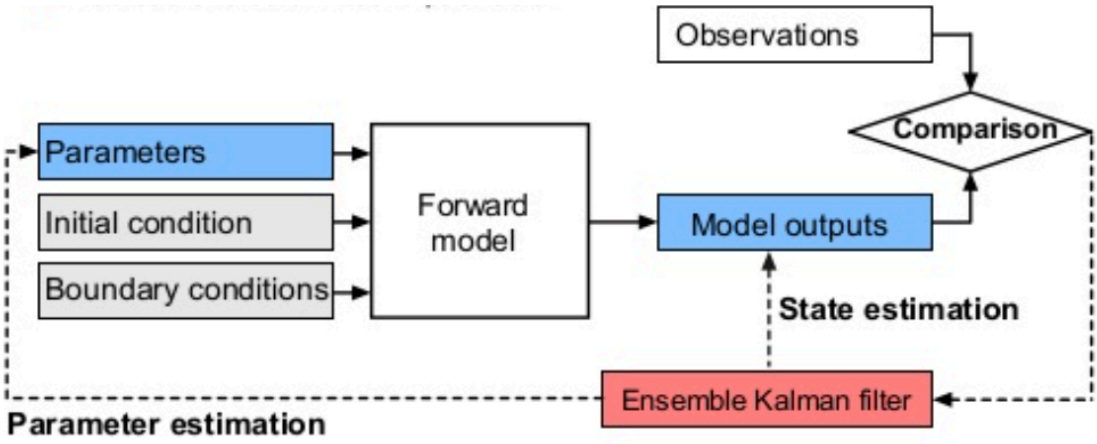

Figure 1.2 The procedure of the EnKF in a flowchart. Image by Rochoux.

1.2.2 Wildfire Data Assimilation

Data assimilation has been proven to be useful in the context of weather forecasting, oceanography prediction, and, in recent years, wildfire propagation modeling. Using the EnKF methodology, scientists have been able to successfully and accurately predict the movement of wildfires on a 4m by 4m scale with the development of Firefly-EnKF (Rochoux, 2014). In addition, WRF-SFire is a data assimilation model that has successfully paired a wildfire simulator with an atmospheric dynamics simulator, developed by Jan Mandel at al. in 2009. State variables in this field include wind speed, wind direction, topography, fuel type, moisture content (in the air and of the vegetation), cloud cover, and canopy height. The multiscale physics of a wildfire are affected by nonlinear interactions with other natural processes, which pose a complex challenge for data assimilation. There may be 5 orders of magnitude of distance to resolve ranging from hundred-kilometer scale weather patterns, meter scale eddies and flame lengths, to centimeter or less scales of combustion and chemical interactions of pyrolysis. Time scale variations also exist, including firelines that may travel with a velocity on the order of meters per second whereas thermal decomposition occurs in seconds or less. The rate of spread (ROS) is determined by the change in position of the fireline normal to itself over the time it takes to move to the new position. Regional scale zone models simulate a 2D fireline that self propagates based on fire dynamics of fuel combustion combined with environmental influences such as wind velocity and topography.

Chapter 2: FARSITE

2.1 Overview of FARSITE

FARSITE, or Fire Area Simulator, is an operational computer simulation growth model for wildfires developed by research scientist Mark Finney in Missoula, MT in 1998. It computes wildfire behavior using physical equations of fire movement combined with spatial and temporal data on weather, topography, and fuel (Finney, 1998). The deterministic software allows the user to analyze the movement of past fires or to predict the propagation of a possible future fire scenario. FARSITE is widely used by federal land management agencies such as the US Forest Service, National Park Service, and others to simulate the spread of wildfires for better understanding resource use and danger implication.

There exist two open source software downloads of FARSITE: an operational-level Windows version and a research-level Linux version Dynamic Link Library (DLL) installation. Both versions of the program are utilized for different purposes in this project. The DLL, run through the terminal with a set of commands, can be downloaded here: http://sbrittain.net/Farsite/Distrib/Linux/Farsite_Linux.htm. It works by calling 3 files: ignition, landscape, and a detailed input file. This procedure will be explained in the following section. The Windows version is the graphical user interface (GUI) for FARSITE and can be downloaded here:

http://www.firelab.org/document/farsite-software. With this application, the user can define inputs and run fire scenarios to visualize the fireline and also extract important data on fire perimeter, rate of spread, and more.

A comparative review of wildfire simulators by Papadopoulos and Pavlidou (2011) states that FARSITE is considered to be the most precise fire propagation simulation model by most researchers around the world. For this reason, its capabilities will hereby be further explored in context with data assimilation. FARSITE model performance has been validated in comparison with several past fires (Finney, 2000). The Rogge wildfire that occurred in the Sierra Foothills in 1996 and burned 22,000 acres is depicted below in Figure 2.1. The observed fire perimeter data in the top image is hand drawn information that has been gathered from direct observation in the air or on the ground. The bottom image is the FARSITE simulation of the same fire, using archived data to provide inputs on wind, topography, and fuel load from the day of the fire initiation. The two fire perimeters are similar in location and successfully support that FARSITE has potential to be a satisfactory physical foundation for experimenting DA wildfire techniques.

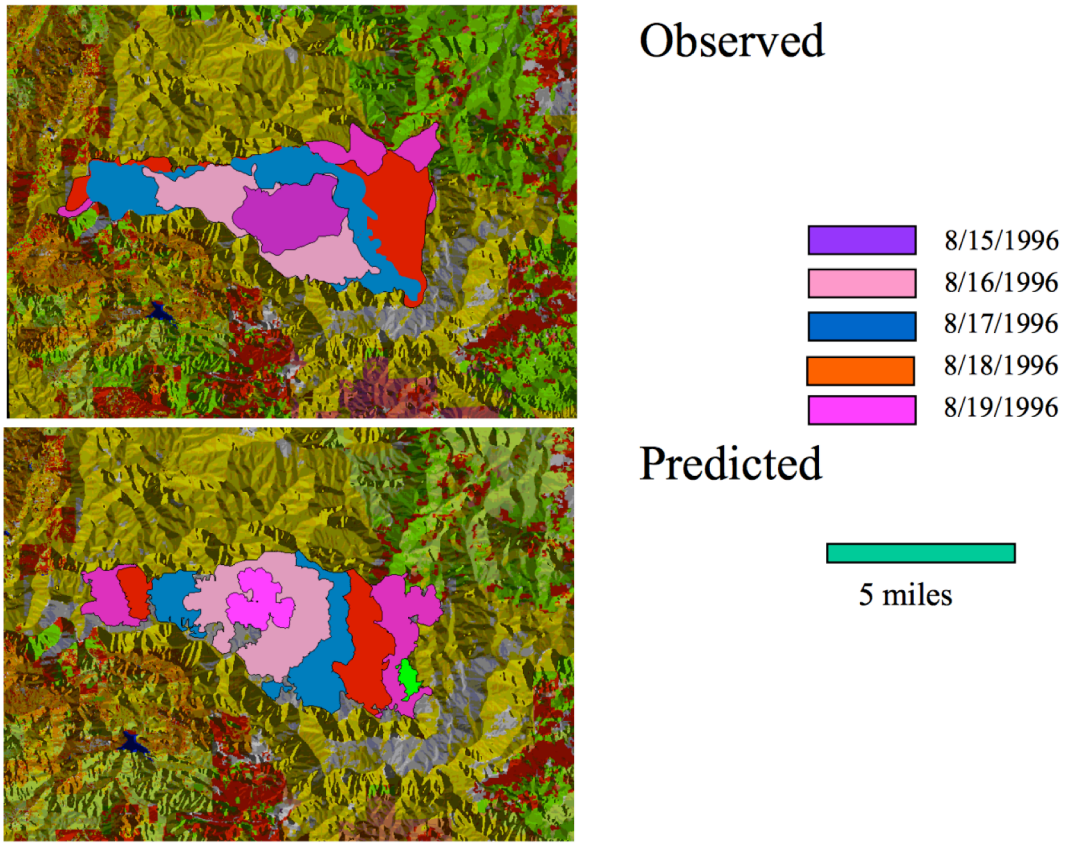

Figure 2.1 Rogge wildfire of 1996 in the Sierra Foothills. The top image is observed fire perimeter and the bottom image is FARSITE simulation (Finney, 2000).

2.1.1 Physical Model

FARSITE uses a 2-D spatial methodology that is based on a Lagrangian 1-D physical model mapped onto a 2-D grid, ideal for modeling fires that are on a moderate size scale. A smaller fire would have critical finite physics that cannot be analyzed in the program and a much larger, intense fire may produce fire phenomena

that, too, cannot be analyzed. The backbone of the 1-D physical model is Rothermel's classical description of surface spread rate of head-fire (Equation 2.1) (Rothermel, 1972). This model was the first to be used by the Forest Service in 1972 as a quantitative, systematic tool for predicting wildfire spread, and remains to be the basis of many models today (Wells, 2008).

$$R = \frac{I_R \xi (1 + \phi_w + \phi_s)}{\rho_b \varepsilon Q_{ig}} \qquad \textbf{(2.1)}$$

Where R = head fire rate of spread [m/min]

I_R = reaction intensity [kJ/min m^2]

ξ= propagating flux ratio

ϕ_w= wind speed coefficient [dimensionless]

ϕ_s= slope coefficient [dimensionless]

ρ_b= ovendry bulk density [kg/m^3]

ε= effective heating number [dimensionless]

Q_{ig}= heat of pre-ignition [kJ/kg]

Assumptions of Equation 2.1 include a simplified wildfire scenario in which the terrain is uniform and the fuel complex is homogeneous. Supporting models that describe more complex features of a wildfire are included in the software, such as

crown fire initiation (Van Wagner, 1977), crown fire spread (Rothermel, 1992), post-frontal combustion (Albini et al., 1995; Albini and Reinhardt, 1995), and dead fuel moisture (Nelson, 2000). The semi-empirical FARSITE model also incorporates an assumed local ellipsoidal fireline shape to produce a more accurate model of the flank fires or rear fires.

Huygen's principle is a vector or wave approach to fire growth modeling, as opposed to a cellular model (Finney, 1998). Essentially the inverse of the cell method, Huygen's principle propagates the fire front at each time step as a continuously expanding fire polygon in order to achieve a 2-D spread (Anderson et al., 1982). The fireline is defined as a set of two vertices (X, Y) and expands to many vertices over time as the fire spreads. The expansion is calculated based on the time step duration multiplied by rate of spread and direction from each vertex. Shown in Figure 2.2, graphic (A) displays uniform conditions of the simulation create symmetrical wavelets with constant shape and size at each time step, forming an even elliptical shape. Graphic (B) displays how non-uniform conditions of the expanding fireline are dependent on wavelet size on the local fuel type, and wavelet shape and orientation on the local wind-slope vector. This creates a more realistic, uneven fireline. Rothermel's model (Eq 2.1) only accounts for the head rate of spread, which is why the mathematical properties of the assumed ellipsoidal shape is used to account for the flank spread.

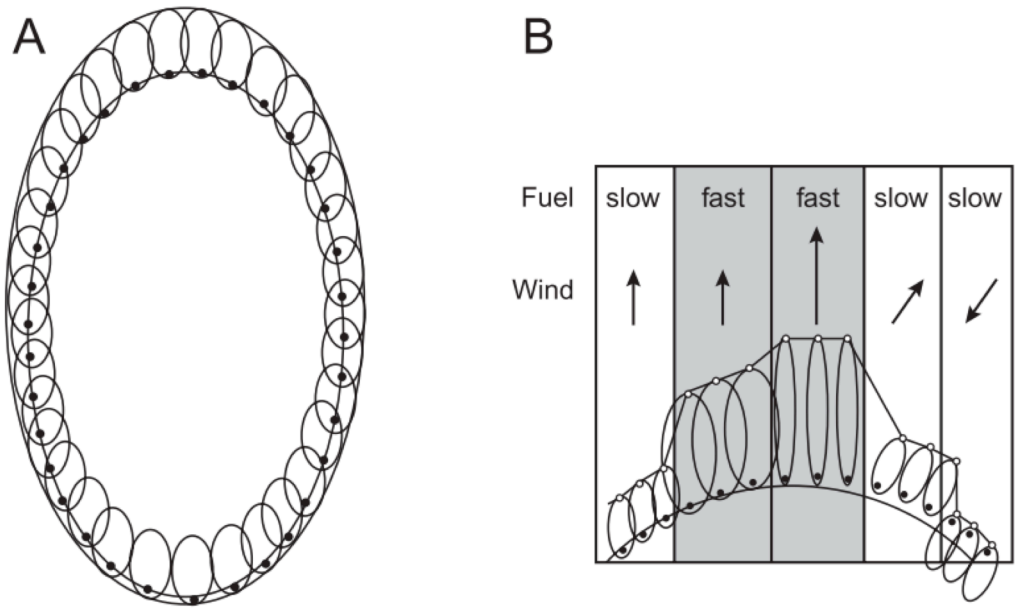

Figure 2.2 Huygen's principle illustrated. (A) Uniform conditions create symmetrical wavelets at each time step. (B) Nonuniform conditions create uneven wavelets at each time step (Finney, 1998).

2.2 Simulation Inputs

2.2.1 Input Files

The FARSITE DLL is used in order to efficiently run multiple fire scenarios with variable wind speed and direction inputs. The Windows version requires more time by the user to input and post process data. There are three input files required to run simulations on the terminal with FARSITE DLL. The process to obtain these files requires a few steps as follows. First, an ignition file, which is formatted as a shapefile (.shp), is created using the GUI Windows software. Shapefile format is a

geospatial vector data format that is used for geographical system information (GIS) software. It stores information on location, shape, and attributes of geographical features. The ignition shapefile created in Windows FARSITE is then exported for use with the DLL. Two additional supporting files are automatically downloaded with the .shp file. These are the shape index format file (.shx), positional index of the feature geometry, and the attribute format file (.dbf), columnar attributes for each shape. The files holds data on where the actual or, if no exact coordinates are predetermined, predicted ignition of the fire occurred. If the user is versed in creating shapefiles via a text editor or graphical information systems tool (i.e. ArcGIS), then the Windows GUI is not necessary.

The next step is to create the landscape shapefile (with extension .lcp), similarly to the ignition file, in the Windows GUI. The landscape file holds critical information on the domain of the simulation, the fuel type, and the fuel moisture content. Once created in the GUI, this file can be exported and used in the input directory for the DLL. Another method of creating the landscape file is to download the .lcp file directly from an online national geo-spatial database called LANDFIRE (Landscape Fire and Resource Management Planning Tools), managed as a shared program between the U.S. Department of Agriculture Forest Service and the U.S. Department of the Interior. This .lcp file can also be altered per user definitions in Windows GUI and then exported as explained above for use in the DLL.

The final input file required is a text file that can be read and edited directly in the terminal. Shown in Figure 2.3, it contains information on time step, weather, wind, and fuel model. Table 2.1 is a list of the mandatory inputs that can be toggled in this file. The "RAWS" (remote automated weather station) matrix columns show the weather input at each time step, including year, month, day, time, temperature, humidity, precipitation amount, wind speed, wind direction, and cloud cover. A MATLAB code can be used that allows the user to automatically change the "RAWS" matrix inputs without ever opening the actual input file. This code found in Appendix A.1 is useful for simulations that take place over many time steps, making it a tedious task to change, for example, the entire wind speed vector. Once the input files contain the necessary, relevant information of the desired fire scenario, the simulation can be run in the terminal with a one-line command: ./TestFARSITE ./Panther/runPanther.txt.

fireflux.input

```
1  FARSITE INPUTS FILE VERSION 1.0
2  FARSITE_START_TIME: 05 03 1200
3  FARSITE_END_TIME: 05 03 1220
4  FARSITE_TIMESTEP: 2
5  FARSITE_DISTANCE_RES: 10.0
6  FARSITE_PERIMETER_RES: 10.0
7  FARSITE_MIN_IGNITION_VERTEX_DISTANCE: 1.0
8  FARSITE_SPOT_GRID_RESOLUTION: 15.0
9  FARSITE_SPOT_PROBABILITY: 0.00
10 FARSITE_SPOT_IGNITION_DELAY: 0
11 FARSITE_MINIMUM_SPOT_DISTANCE: 30
12 FARSITE_ACCELERATION_ON: 1
13 FARSITE_FILL_BARRIERS: 0
14 SPOTTING_SEED: 253114
15
16
17 FUEL_MOISTURES_DATA: 1
18 0 3 4 5 30 60
19 RAWS_ELEVATION: 0
20 RAWS_UNITS: English
21
22 RAWS: 5
23 2013 5 3 1140 68 30 0.00 10 10 0
24 2013 5 3 1150 68 30 0.00 10 10 0
25 2013 5 3 1200 68 30 0.00 10 10 0
26 2013 5 3 1210 68 30 0.00 10 10 0
27 2013 5 3 1220 68 30 0.00 10 10 0
28
29 FOLIAR_MOISTURE_CONTENT: 50
30 CROWN_FIRE_METHOD: Finney
31 NUMBER_PROCESSORS: 1
```

Figure 2.3 Sample input file for the FARSITE DLL.

Table 2.1 Mandatory switch inputs for FARSITE DLL.

Line #	Name	Description
1	FARSITE INPUTS FILE VERSION	File version number
2	FARSITE_START_TIME	Month, Day, Hour, Minute
3	FARSITE_END_TIME	Month, Day, Hour, Minute
4	FARSITE_TIMESTEP	Actual Time Step in minutes
5	FARSITE_DISTANCE_RES	Distance Resolution. FARSITE will check for new fire characteristics when this distance has been covered within a time step.
6	FARSITE_PERIMETER_RES	Perimeter Resolution. Fire vertices every X meters along a perimeter while burning t each time step.

Table 2.1 Mandatory switch inputs for FARSITE DLL.

Line #	Name	Description
7	FARSITE_MIN_IGNITION_VERTEX_DISTANCE	The minimum distance between vertices in an ignition.
8	FARSITE_SPOT_GRID_RESOLUTION	Resolution of the background spotting grid.
9	FARSITE_SPOT_PROBABILITY	Represents the probability that an ember can survive to intersect the landscape.
10	FARSITE_SPOT_IGNITION_DELAY	Represents the delay time in minutes before a spot fire is started after an ember lands on a burnable substrate.
11	FARSITE_MINIMUM_SPOT_DISTANCE	The distance an ember must travel before it can start a spot fire.
12	FARSITE_ACCELERATION_ON	If on, accelerates the rate of spread.
13	FARSITE_FILL_BARRIERS	Where X is either 0 for false (no barrier fill) or 1 for true (fill the barriers). FARSITE DLL will set all of the pixels inside a barrier polygon to non-burnable.
14	SPOTTING_SEED	The seed to be used to initialize the random number generator for spotting.
17	FUEL_MOISTURES_DATA	Default moistures to use when a fuel model is encountered in the lcp file that does not have an entry in the inputs file.
18	Fuel Moistures Data Matrix	Model, FM1, FM10, FM100, FM Live Herb, FM Live Woody
19	RAWS_ELEVATION	The elevation of the RAWS data in feet or meters.
20	RAWS_UNITS	Units English or Metric
22	RAWS	The number of RAWS weather entries in the succeeding matrix.
23-27	RAWS Matrix	Month, Day, Precipitation, Minimum Temperature hour 0-2400, Maximum Temperature hour 0-2400, Minimum Temperature, Maximum Temperature, Maximum Humidity, Minimum Humidity, Elevation, Precipitation Start Time, Precipitation End Time
29	FOLIAR_MOISTURE_CONTENT	Default 100% Moisture Content
30	CROWN_FIRE_METHOD	Finney or Scott Rhienhardt
31	NUMBER_PROCESSORS	The number of processors used to run the simulation.

2.2.2 Wind Data

Wildfire spread and growth depends on three primary physical parameters: wind, topography, and fuel characteristics. On a contained time scale that spans the duration of a wildfire, topography and fuel load at a specific location are measurable and static. Wind, however, may change at every second, every day in terms of speed and direction. In addition, large wildfires have the ability to produce their own wind field, which further affects the dynamics of fire spread and growth. Any of the above input files containing these physical parameters may be altered per the simulation requirements, but of most interest and of most consequence are the wind variations.

Observational wind data for wildfires is collected at the height of the weather instrumentation, the US standard for RAWS being 20 feet above bare ground or vegetation (Andrews, 2012). Wind data collection for structure fires follows alternate standards. To account for spatial differences in wind speed, FARSITE automatically calculates a midflame wind velocity using the provided observation data and a wind adjustment factor correlation (WAF). The "effective" midflame wind speed, relevant in the context of Rothermel's model, encompasses the combined effect of slope and wind on head fire spread rate (Finney, 1998). It is not specifically calculated at the mid-height of the flame, however, the term is used to distinguish between the free wind above vegetation and the reduced wind that is used in Rothermel's equation (2.1). There are two WAF correlations, one for unsheltered wind speed (no overstory) (Eq. 2.2) and the other for sheltered wind speed (with overstory vegetation)

(Andrews, 2012). For this project, only unsheltered conditions were examined and thus sheltered correlations will not be considered. To calculate the midflame wind speed given the WAF equation 2.2, multiply the WAF by the 20-ft wind speed, producing a new wind value in miles per hour. Figure 2.4 displays how the midflame wind speed changes depending on the fuel bed depth (H) for the unsheltered case, given a 20-ft wind speed of 15 mph.

$$WAF = \frac{1.83}{\ln\left(\frac{20 + 0.36H}{0.13H}\right)} \quad \textbf{(2.2)}$$

Where WAF = Wind Adjustment Factor

H = Fuel Bed Depth [ft]

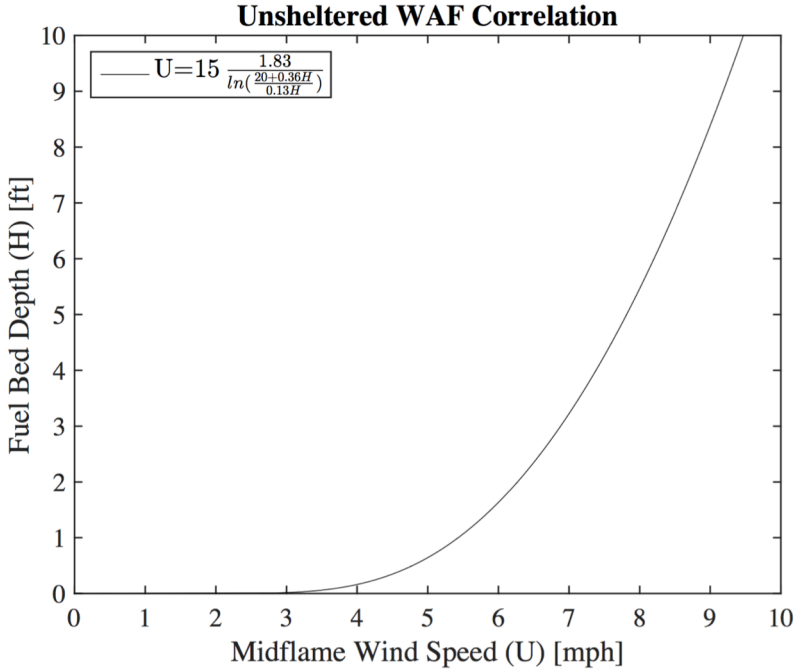

Figure 2.4 Given a 15 mph wind speed at 20-ft height, the midflame wind speed is dependent on the fuel bed depth.

2.3 Limitations of FARSITE

Although FARSITE is the most commonly used and accepted wildfire simulation program, it bears limitations that are important to consider. One notable limitation is the restriction on the domain size. The program is intended to be used with large-scale fires and will not accept a domain smaller than 30 m by 30 m, based on landscape compatibility constraints. This makes it impossible to resolve small-scale data, finite physical interactions, and near-surface dynamics. In addition, the program is not capable of modeling extreme fire behavior such as fire whirls or convection columns. If a fire is categorized as conflagration, it is possible that it produces its own fire-induced winds, which is not modeled with FARSITE.

Limitations of the program also arise from the real-world topography data that is extracted from the online government database LANDFIRE. The user is restricted to the landscape details provided within those data files. The landscape may change gradually due to erosion, drought, wildlife habitation, etc., or it may change very rapidly due to a fire, man-made deconstruction, or construction. The LANDFIRE data is only updated every 3 years, causing a lot of these changes to go unreported in the provided landscape file and thus the wildfire simulation is not as accurate as possible. This can affect community planning or firefighter preparedness. Another issue with the LANDFIRE files is that the wildland-urban interface (WUI) by definition means that there are houses and/or urban properties in the potential hazard area. The physics of the program cannot yet incorporate the complexity of a WUI fire in which a

wildfire transitions to a structural fire, or vice versa, and how the two interact and "fuel" each other. The landscape file that may be used from LANDFIRE will consider structures as "non-burnable area", which is not the case in real life.

Finally, the limitation that poses the most accuracy concerns is that the current version of FARSITE DLL does not use a spatialized wind field. The wind is assumed to be uniform at any height across the given topography. This will lead to inevitable error because wind is a dynamic field that varies over terrain, vegetation, and even with height above the ground. However, with the small-scale experiment examined within this paper, the terrain and vegetation are uniform meaning that the wind need not be spatialized across the domain. In addition, static wind speed versus height above the ground is accounted for within the physical model using a Wind Adjustment Factor (WAF) as explained in the previous section.

Chapter 3: FARSITE with Data Assimilation

3.1 Overview of the Technical Applications

Integrating the physical model, FARSITE, with the statistical model, EnKF, requires a third party dynamic code coupler called OpenPALM, developed by CERFACS (Centre of Basic and Applied Research Specialized in Modeling and Numerical Simulation) in Tolouse, France. OpenPALM allows for two programs to run and communicate with each other simultaneously based on a combination of application-provided and user-written code. The capabilities of a dynamic coupler such as OpenPALM include data exchange, intermediate computations, grid-to-grid interpolation, and parallel data redistribution (Piacentini, 2003). These functions are possible through the libraries that are provided with the coupler program installation. PALM is a dynamic algebraic toolbox library that allows the application to exchange data, parallel compute, and redistribute the information to the appropriate component. CWIPI (Coupling with Interpolation Parallel Interface) is a static library that incorporates mesh-based coupling in 1D, 2D, or 3D exchange zones that can be discretized according to the user demands. CWIPI connects the communication gaps between the different meshes.

There are two main procedures for reconstructing the FARSITE DLL to use with DA. The first part is to access a super computer cluster and install the parallel

coupler program, OpenPALM. The second part is to integrate FARSITE within OpenPALM for use with the EnKF algorithm. For large, parallel jobs such as this, the University of Maryland provides students and faculty with a supercomputer called Deepthought2 (DT2) High-Performance Computing (HPC). DT2 is powerful enough to concurrently run multiple simulations over many processors. Detailed instructions for accessing this HPC and completing these implementation procedures are listed in Appendix B.1. Downloading and using OpenPALM on DT2 requires the installation of a Secure Shell (SSH) client. Once the application is downloaded, the OpenPALM environment variables must be configured according to the DT2 pathways, as explained in Appendix B.2. After the setup is complete, jobs can be run through the terminal on DT2 using a queue system designed to evenly allocate CPUs amongst its users. Running a simulation requires model parameters to be defined in the input files and placed in the correct directory path. The easiest way to modify an input file before a simulation run is to download a secure FTP client, such as Fetch or FileZilla, so the user has access to transfer files to the remote HPC.

3.2 Technical Details of the FARSITE-OpenPALM Application

The parallel implementation of FARSITE with EnKF completed with OpenPALM allows data assimilation by intercommunication between the programs to optimize the simulation outputs. The goal is to simulate the fireline at pre-specified time intervals given the initial conditions and use the inverse loop to recover from the

propagation errors. The EnKF works by making amplitude corrections in wind speed or direction, rather than position corrections (Mandel et al., 2009). The physical model takes in the error correction and generates a new fireline based on a probabilistic best fit for each ensemble member. These action items must be user-coded and are not pre-loaded within the OpenPALM environment.

The GUI of the parallel application, PrePALM, allows for visualization of the EnKF and a better understanding of how it communicates to the different components within itself. It is used to enter and control the many units of a program that need to cooperate. This section will further explain how the framework of the application is laid out and what it is that the GUI presents. The application shown in Figure 3.1 is a single unit program on one branch, in this case FARSITE. The sequence of elementary actions within a unit follows the logic of a programming language in which contains declared variables, instructions, and control structures (loops and conditional switches) (Piacentini, 2003). If there are multiple units in an application the PALM driver interprets in run-time the code and schedule of execution for each unit. Figure 3.2 displays the FARSITE ensemble application with three branches that hold four units. The branches are in task level parallelism and the blue branch displays internal parallelism; parallel in this context meaning the tasks are divided into a number of processors to simultaneously carry out their functions. More specifically, the first unit of the blue branch manages the FARSITE program and parameter inputs. It tells the red branch to run the simulation. The red branch contains 1 master and 4 slave processors that run simultaneous integrations of the forward

model. The outputs from the red branch go back into the blue branch and are now inputs for the blue ensemble unit. This is where the ensemble members are created for data assimilation. The green branch receives information from the ensemble unit and slave processors to create a simulated front data ensemble.

The next step of the algorithm is the observation comparison to prior forecast ensemble data from the green branch. In this step, a background covariance error matrix is produced based on user defined standard deviation and mean ensemble output. The observation comparison and error is then used to update state and posterior parameter estimation through the inverse model. One challenge with the FARSITE inverse model is that the Lagrangian method of fire propagation creates an inconsistent number of "markers" that characterize the fire front for each ensemble member and at each time step. When the analysis and inverse update steps occur, the number of markers in the fire front needs to be identical for proper parameter and state comparison. To overcome this dysfunction, a defined number of markers are chosen to assimilate each front similarly. The algorithm is coded to produce 100 markers for each front. If the front has fewer than 100 markers, the space in between the markers is interpolated to produce more points. If the front has greater than 100 markers, the excess amount is discarded.

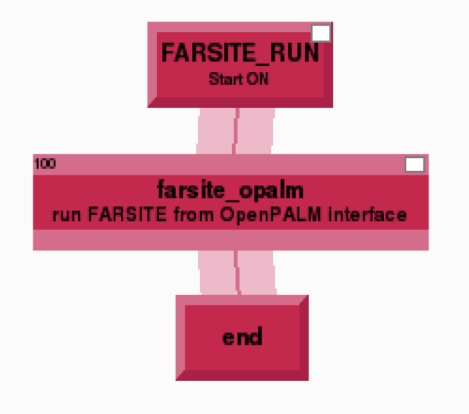

Figure 3.1 A single-unit program displayed in the OpenPALM interface. This unit is the FARSITE DLL and is capable of running a deterministic simulation.

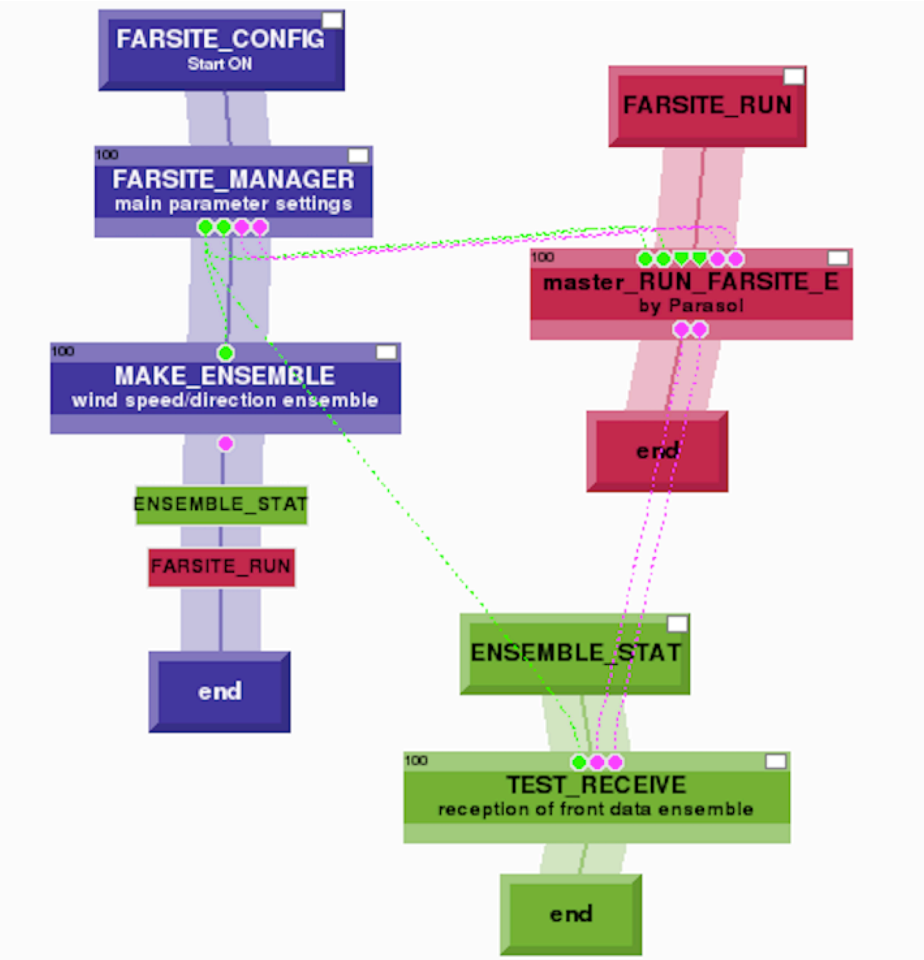

Figure 3.2 A four-unit program displayed in PrePALM. The blue units manage the FARSITE application, the red unit holds the master and slaves, and the green unit receives the simulation data. This configuration is capable of running ensemble members of a FARSITE simulation.

3.4 Post Processing Data

Post processing the simulation data is done in two different but similar ways, outside of the DT2 environment and within the DT2 environment. Outputs can be extracted in spatial, linear, and tabular forms, increasing the options for analysis. With the standalone FARSITE DLL outside of DT2, the output data is dumped into an output folder in the form of many different files, each containing important data for post processing. To plot the simulated fireline, the shapefile "Perimeters.shp" can be opened in MATLAB using functions "shaperead" and "geoshow" available with the MATLAB mapping toolbox. This is useful for mapping a predicted fireline over an observed fireline and comparing the differences. If the two lines are very dissimilar, this tells the user that the simulation is far off from actual data. The file "SpreadRate.asc" can be used to determine the maximum rate of spread for a simulation. This is an ActionScript Communication (.asc) text file that contains an array of data, of which the highest value is the maximum ROS. The array can be copied into MATLAB and the function "max" is an easy way to find this maximum value. Other information that can be extracted in post processing includes spotting outcome, heat per unit area, flame length, and crown fire ignition. This data is secondary to determining how well simulation matches observation and therefore was not analyzed in the scope of this project.

Within the DT2 environment, the same output files are received as the standalone FARSITE. However, a streamlined process for plotting the fireline has

been incorporated into the DA OpenPALM scheme. The program is coded in such a way that data points are extracted as (X,Y) coordinates at each time step, making it straightforward to plot the fireline within MATLAB. This MATLAB code is included in Appendix A.2.

Chapter 4: Model Demonstration with FireFlux I

4.1 Overview of the FireFlux I Experiment

One of the major difficulties of validation of wildfire forecasting models is the lack of observational data available to the scientific community. There is high possibility of endangering personnel and damaging instrumentation in order to collect the necessary observations. Aerial sensors and infrared technology are other methods of collecting data, but these are expensive and still cannot capture all the observations needed for analysis. Initial attempts at overcoming this challenge include setting up prescribed, controlled burn trials with intent to collect physical data and measurements. These trials are typically less intense than an accidental wildfire, but are sufficient to provide a basis for model validation. FireFlux I is the first field-scale wildfire experiment conducted by Craig Clements of the Fire Weather Research Laboratory to "simultaneously measure fine-scale atmospheric circulations, turbulence structure, and plume thermodynamics" (Clements et al., 2007). The experiment, performed in 2006 at the Houston Costal Center, remains one of the most comprehensive grass-fire experiments to this day.

The FireFlux I experiment was conducted on a 0.63 km^2 flat terrain plot of homogeneous, tall-grass prairie land that is maintained by annual mowing. Specific vegetation in the plot was big bluestem (*Andropogon gerardi*), little bluestem

(*Schizachyrium scoparium*), and long spike tridens (*Tridens strictus*) (Clements et al., 2007). Table 4.1 outlines the input parameters for the ROS model that FARSITE uses in its input files. The wind speed input data was collected from Clements and Kochanski, who provided an analysis of the experiment using the model WRF-SFire. Anemometers were set up at 2 meters high, 10 meters, 20 meters and 43 meters on an instrumented tower. Data from these instruments reveal that the wind speed varied from 3 m/s near the 2 m height anemometer to 7 m/s near the 43 m height anemometer. Because FARSITE automatically adjusts a 20 ft (6 m) wind speed to a wind speed at midflame height, and the observation data showed that wind speed changes drastically around the fire front, the wind speed input chosen to run the simulation is 4.5 m/s (10 mph). The rate-of-spread of the fire was determined to be approximately 1 m/s for the 15-minute duration of the experiment.

Table 4.1 Fuel and wind input parameters for the FireFlux I experiment.

Fuel and Wind Parameters for FireFlux I	
Fuel Loading	1.08 kg/m^2
Fuel Depth	1.5 m
Fuel Particle Density	400 kg/m^2
Fuel Surface-to-Volume Ratio	5000 m^{-1}
Dead Fuel Moisture Content	9%
Heat of Combustion	15.4 MJ/kg
Wind Speed (20-ft)	4.5 m/s
Wind Direction	10^o south-west

While the experiment instruments recorded sufficient data for analysis of the smoke plume, one limitation is that fireline information with suitable spatial and temporal resolution was not captured. However, Jean-Baptiste Filippi at the University of Corte in Corsica, France was able to produce surrogate firelines using arrival time of the fire front. ForeFire/Meso-NH, the coupled wildfire-atmospheric model that Filippi used to simulate FireFlux I, provides good agreement between numerical result and observation data. Figure 4.1 shows the results from Filippi's simulation where the front is plotted at 120-second time intervals in a domain that covers the size of the experimental burned grass area, 380 m in the x-direction and 790 m in the y-direction. This plot will be used a basis for comparison of FireFlux I observations to the FARSITE-EnKF simulations.

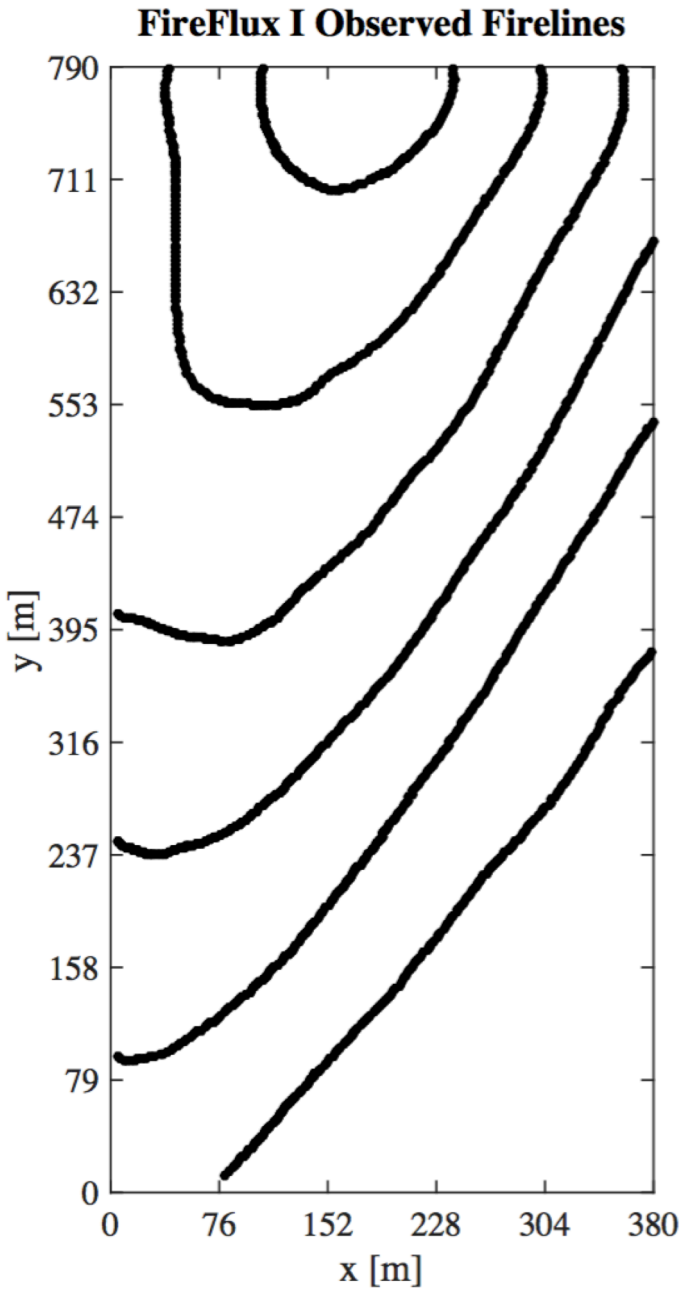

Figure 4.1 Observed firelines of the FireFlux I experiment produced by Filippi using front arrival time data in ForeFire/Meso-NH. The time step between each front is 120 seconds.

4.2 Sensitivity Study

A one-dimensional sensitivity analysis of a "black box" type application is a critical step in model validation to discern how a slight change of inputs will affect the corresponding outputs. In this case, wind speed and wind direction were both studied under a sensitivity analysis using FARSITE DLL and the FireFlux I experiment. The wind speed was varied in increments of 0.447 m/s (1 mph) from 0 m/s to 10 m/s and graphed versus the corresponding rate of spread outcome, as seen in Figure 4.2. Wind direction was plotted using the same fireline ignition for 5 trials at varying degrees of -50º, -25º, 0º, 25º, and 50º, as seen in Figure 4.3. Because ROS is reported in an ASCII grid file format with extension ".asc", MATLAB was used to extract the highest value from this output file. This number is given by default in ft/min, so a metric conversion gives the ROS in m/s, as graphed.

The results of the wind speed sensitivity analysis show that for each increasing unit, the head fire ROS also slightly increases. With zero wind speed input, the fire will not spread in the FireFlux I simulation. Based on Rothermel's model, it is possible the ROS can be nonzero with no wind conditions as long as the other parameters are conducive (dry, dense fuel) for fire spread. The results of the wind direction sensitivity analysis show that 25º increments are enough to significantly change the course of fire propagation. It is important to correctly choose the input parameters as any small deviation in the inputs can cause inaccuracy from the desired outputs. However, initial condition and boundary data is not always available and

must be user-selected, which is why the data assimilation scheme is used to smooth out those estimates and provide a best fit of the parameters.

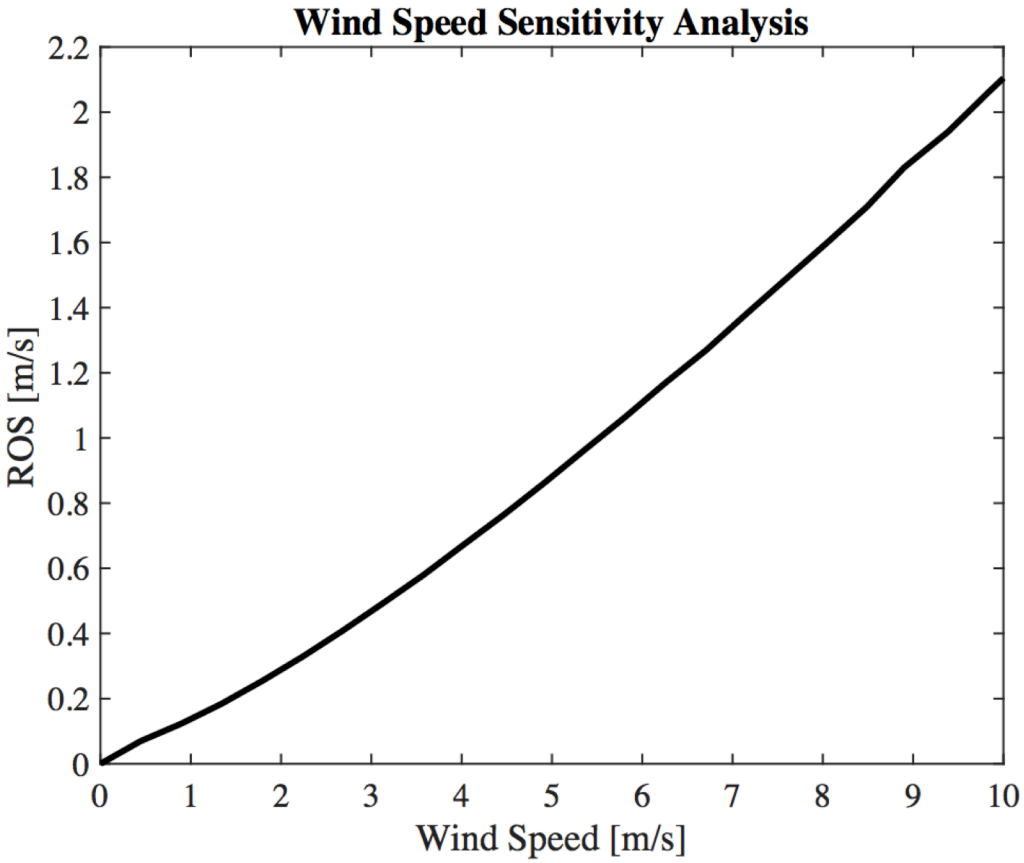

Figure 4.2 Rate of spread for the head fire of FireFlux I versus increasing wind speed in meters per second.

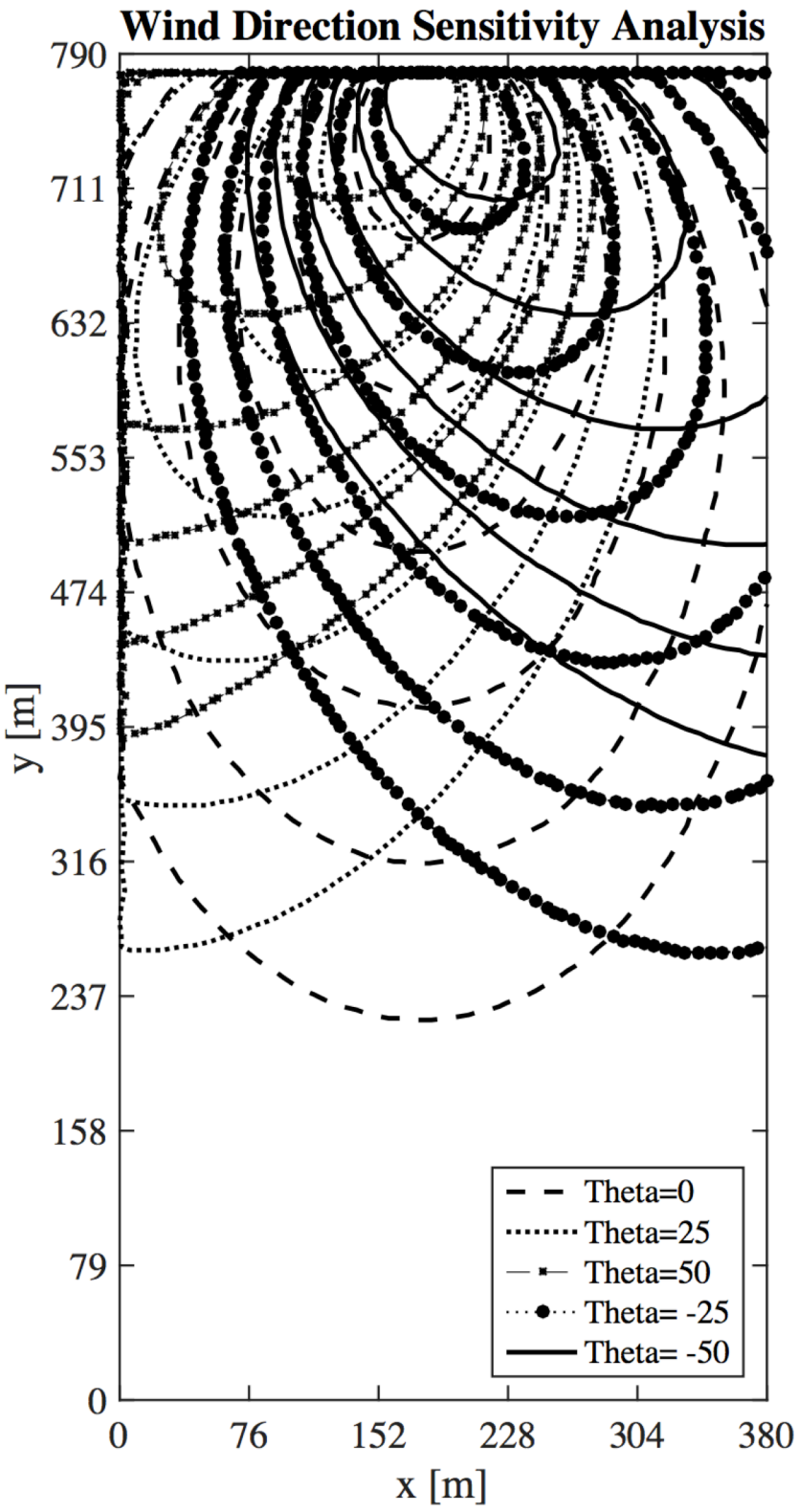

Figure 4.3 Varying wind direction effects on FireFlux I experiment.

4.3 Deterministic Test Case

4.3.1 Overview

Before data assimilation can be utilized for the FireFlux I experiment, a deterministic test case must be run to analyze how the stand-alone physical model FARSITE predicts wildfire propagation within the OpenPALM application, prior to incorporating the statistical EnKF model. The deterministic cause and effect optimization can only be produced for a short- or medium-range prediction. The data produced further out from present time of the observation results in less accurate forecast. The inputs used in the FARSITE application were based on the observation data from the FireFlux I experiment, as given in section 4.1. The deterministic test case is not expected to have high accuracy to the observed data. The high error may derive from the equations within the model not fully incorporating atmospheric physics, the model resolution not sufficient to capture finite thermodynamic processes, and inaccurate or incomplete initial observations.

More so, the deterministic test case is a verification of the forward model using known control parameters in the control vector. The computational domain of the simulation was constructed to be as close to Filippi's observations as possible, but FARSITE is restricted to 30 m grid increments. Therefore, the domain is 390 m long in the x-direction and 780 m long in the y-direction, and the landscape resolution is 30 m. The perimeter resolution and distance resolution are both 10 m. The total time

of the simulation is 720 seconds, with a fire front produced at every time step of 120 seconds.

4.3.2 Results

Figure 4.4 displays the simulated fire front, the dotted line, versus the observed fire front, the solid line. The simulated maximum ROS is 0.8 m/s whereas the observation ROS is 1 m/s. The error is likely based on incorrect initial conditions such as initial ignition coordinates or wind speed input. However, FARSITE is limited to integer wind speed inputs and multiple tries of "guessing" the correct inputs will not produce anything more accurate than what is presented below. Another reason why the firelines do not match well on this plot could be attributed to the 10 m shorter difference in the y-direction domain of the simulated front. The flanking of the fireline in the east and west (x) direction is an improvement to previous forward models that did not incorporate the physics of the elliptical shape of wildfire spread. As expected, the deterministic run of FARSITE does not provide a highly accurate forecast.

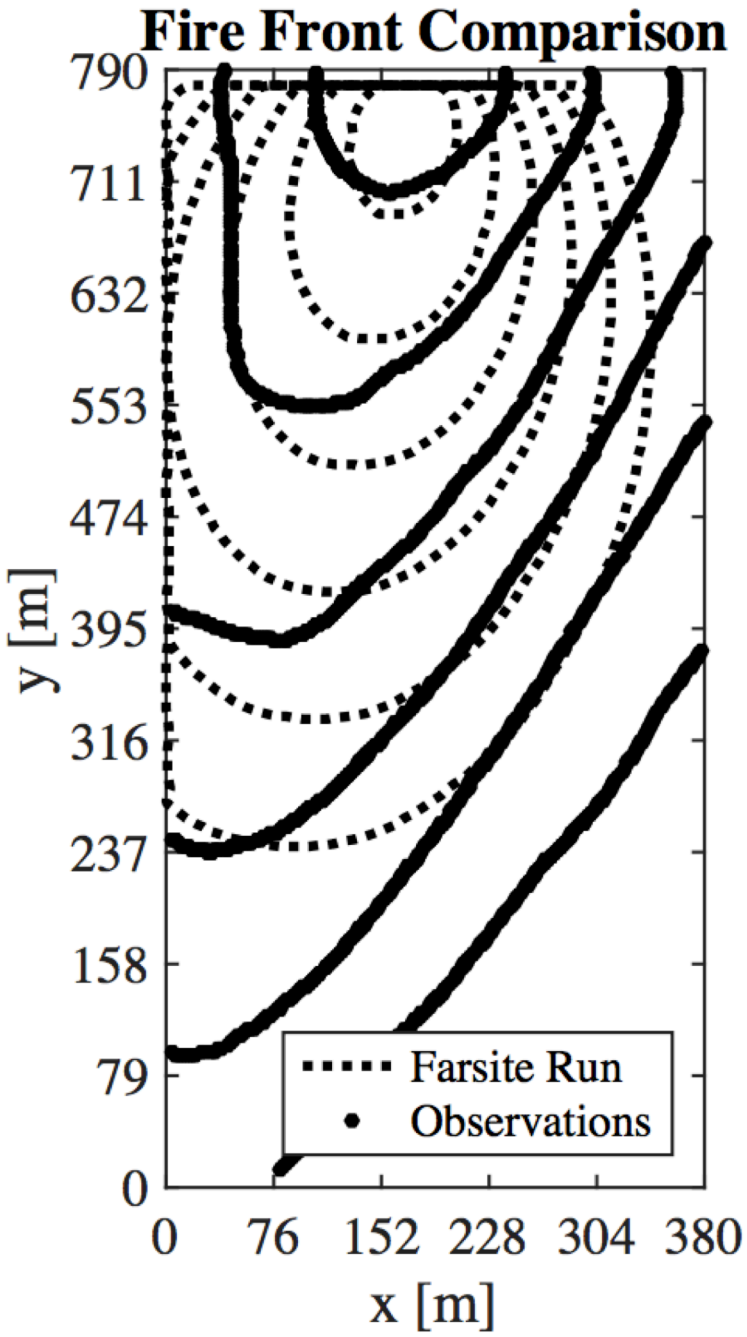

Figure 4.4 Deterministic run of the FireFlux I experiment as simulated by FARSITE and compared to observation data produced by Filippi. The time step between each front is 120 seconds.

4.4 Ensemble Test Case

4.4.1 Overview

Providing an ensemble test case is the first step in the data assimilation method. The algorithm configured in OpenPALM produces 60 ensemble members over 4 slave processors at each time step of 120 seconds. The standard deviation for the wind speed and direction inputs are user defined in the file "wind_ens.input". This tells the system for each ensemble member how much to vary the given input wind speed and wind direction in order to later assimilate the stochastic outputs and produce a best estimate to observation. For the FireFlux I simulation, mean wind speed input is 4.5 m/s with a selected standard deviation of 3 m/s. Mean wind direction input is 10^{o} with selected standard deviation of 4^{o}. The domain is again 390 m long in the x-direction and 780 m long in the y-direction, and the landscape resolution is 30 m. The perimeter resolution and distance resolution are both 10 m. The total time of the simulation is 720 seconds.

4.4.2 Results

The ensemble simulation produced output data for 60 members at 6 time steps. Figure 4.5 presents 8 of 60 ensemble members on the same plot as the deterministic result and observed fireline data at time step 2, 240 seconds. These

members were randomly chosen to be displayed in the plot for readability, rather than plotting all 60 members. The number of markers of each member is based on the size of the fireline, which is dependent on the magnitude of the wind speed. A faster wind speed will produce a fireline that travels farther and has more data points, or marker numbers, to plot. It is clearly visible that the model produced members with varying wind directions (i.e. members 1 and 11) and varying wind speeds (i.e. members 1 and 16). This shows that the ensemble algorithm is working correctly. The stochastic outputs of the ensemble show results that a newly created state estimate at time step 2 will align more closely with observed data than did the deterministic run. The inverse loop of the EnKF, not yet in the algorithm, will recursively run the forward model with an updated parameter estimate and assimilate the members to provide a best-fit state output. Sources of error, defined as distance between the observed and projected front, may come from unknown initial conditions. The goal of data assimilation is to smooth out those uncertainties and provide a better parameter estimate, regardless of how inaccurate the initial parameters are.

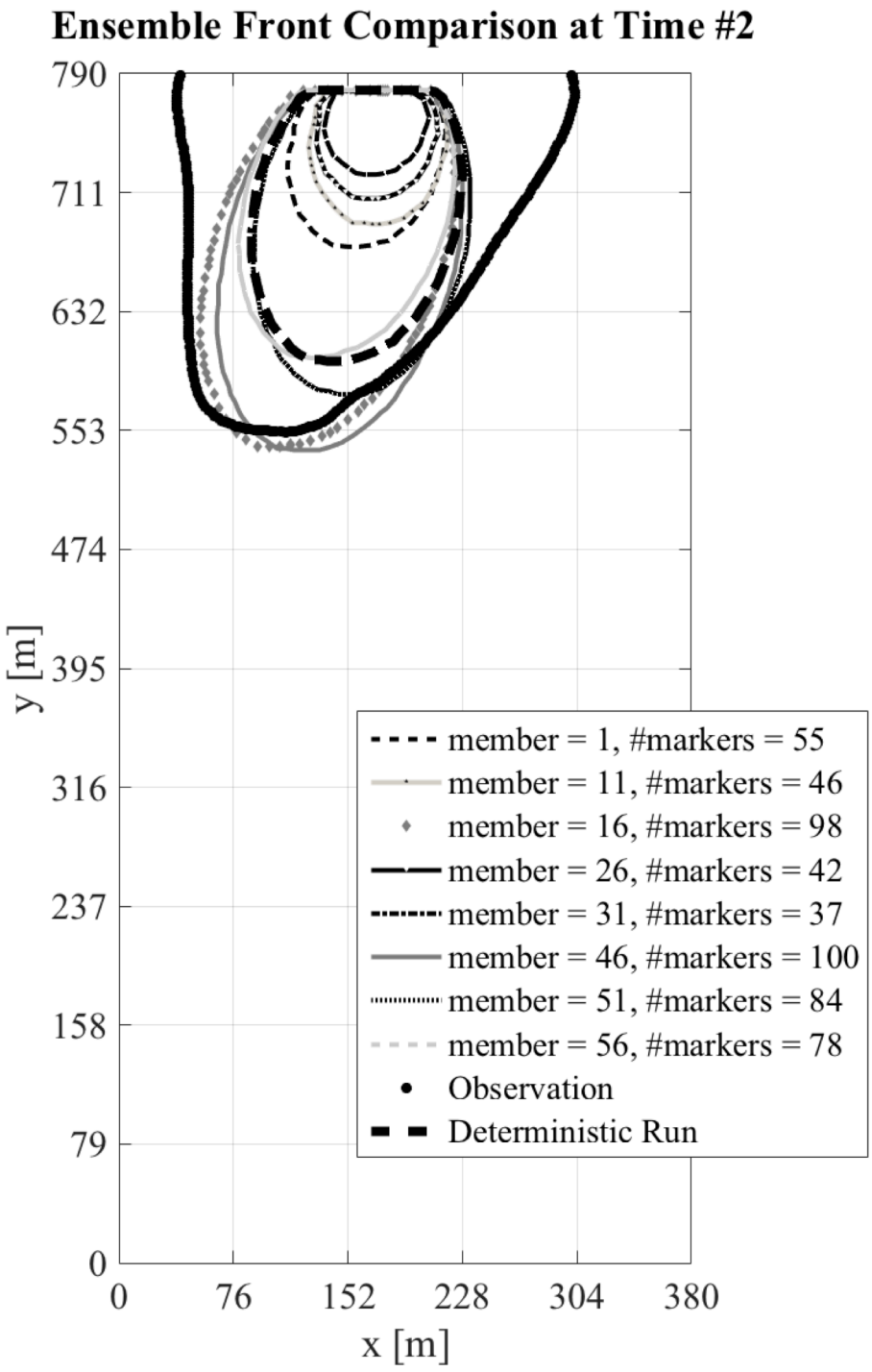

Figure 4.5 Comparison of 8 randomly selected ensemble members to the deterministic result and to the observed fireline data at time = 240 seconds.

4.5 Statistical Model Analysis

The Ensemble Kalman Filter has the best performance when the control vector is generated with Gaussian distribution. Demonstrated in this section is an evaluation that the statistical model is set up correctly in the OpenPALM environment. The following figures of the probability density function (PDF) provide proof that the EnKF is using proper Gaussian distribution for probability of events. The bar graphs display distribution in which the sum of y over all values of x gives a probability value of 1. Figure 4.6 shows the wind speed distribution given a mean value of 4.5 m/s with a standard deviation of 3 m/s. The probability density is plotted versus the various wind speed values that are extracted from each of the 60 ensemble members. Figure 4.7 shows the probability density versus wind direction distribution, given a mean value of 10^{o} and a standard deviation of 4^{o}. The "bell-shaped" curve in both figures is indicative that the statistical algorithm is applied correctly. The distribution would be more similar to a smooth Gaussian distribution in both figures if a greater number of ensemble members were produced.

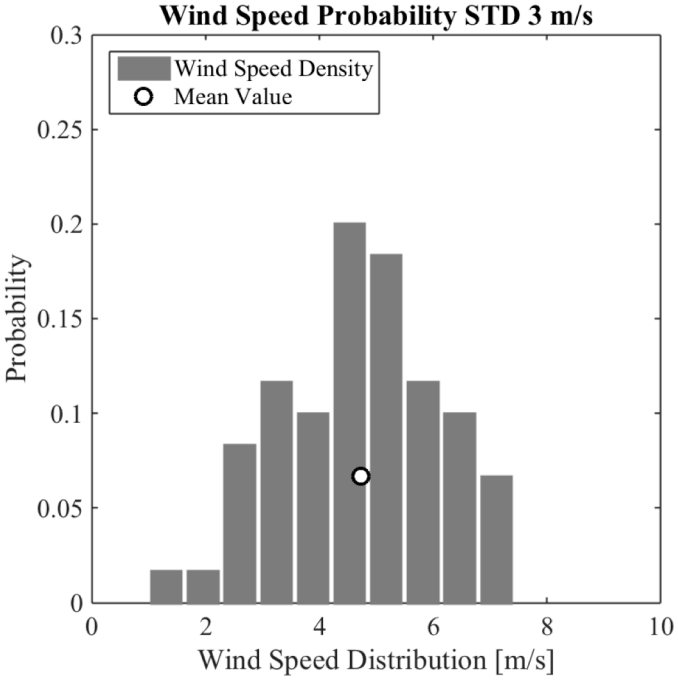

Figure 4.6 Probability density evaluation of wind speed given a mean value of 4.5 m/s and a standard deviation of 3 m/s.

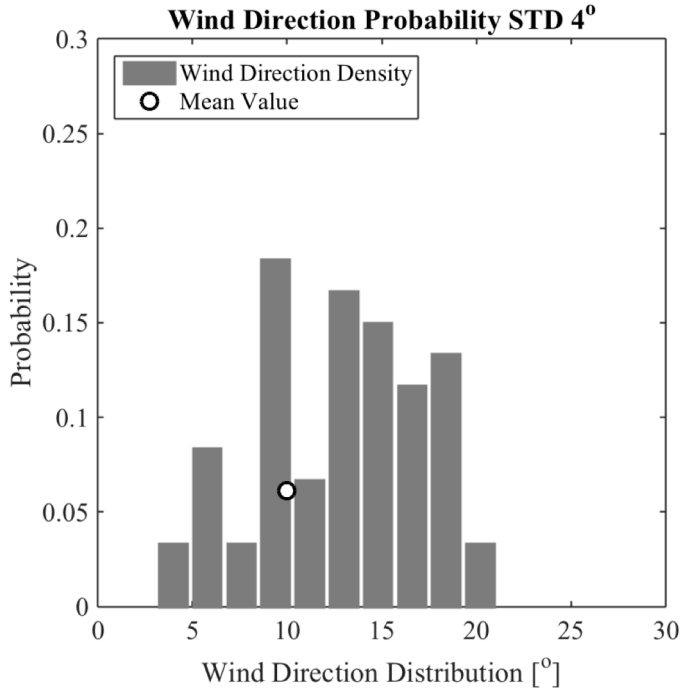

Figure 4.7 Probability density evaluation of wind direction given a mean value of 10^{o} and a standard deviation of 4^{o}.

Chapter 5: Model Demonstration with RxCADRE S5

5.1 Overview of the RxCADRE Experiment

Demonstration of FARSITE-EnKF capabilities continues with another model comparison to RxCADRE S5 experiment. The series of RxCADRE experiments, Prescribed Fire Combustion and Atmospheric Dynamics Research Experiments, were a collaborative effort funded by the Joint Fire Science Program conducted in 2008, 2011, and 2012 at the Eglin Air Force Base in Florida. The purpose of the multiple experiments was to systematically collect measurements on fire behavior, fire effects, smoke chemistry, and dynamics in order to provide data for improving fire model validation. A comprehensive database was compiled using state-of-the-art instrumentation and technology that has never previously been used for fire research (Wells, G., 2013).

The S5 experiment examined here is a small-scale burn with block size of approximately 100 m by 200 m. The fuel burned was a patchy, heterogeneous mix of grass with light shrub. Further details on the fuel characteristics can be found below in Table 5.1. Observed firelines, shown in Figure 5.1 were produced using video and infrared technology to capture images of the flaming fire front. The fire develops flanks at three minutes into the experiment. The maximum ROS of the fire front was 0.25 m/s.

Table 5.1 Fuel and wind input parameters for the Fireflux I experiment.

Fuel and Wind Parameters for RxCADRE S5	
Fuel Loading	0.28 kg/m^2
Fuel Depth	0.2 m
Fuel Particle Density	513 kg/m^2
Fuel Surface-to-Volume Ratio	9000 m^{-1}
Dead Fuel Moisture Content	10%
Heat of Combustion	18.6 MJ/kg
Wind Speed (20-ft)	2.5 m/s
Wind Direction	345° south-east

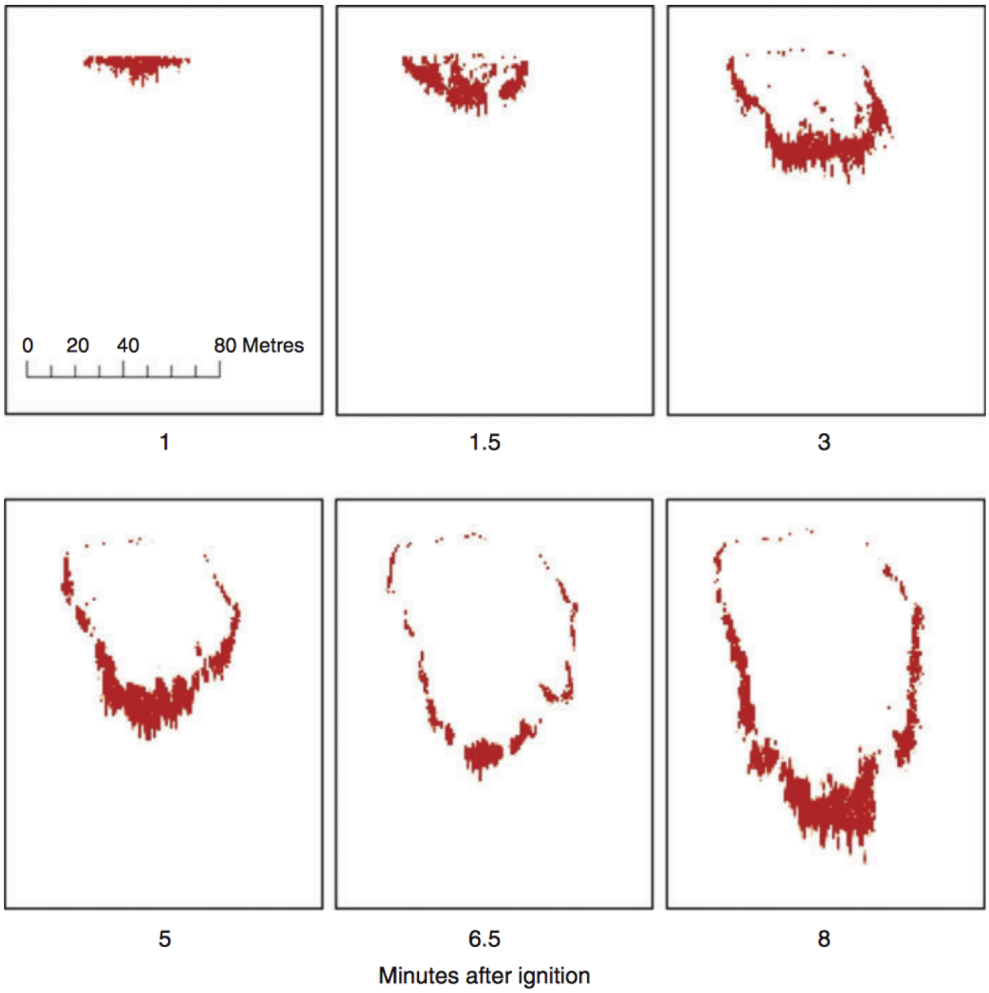

Figure 5.1 Observed firelines of the RxCADRE S5 experiment. The block size is approximately 100 m by 200 m.

5.2 Deterministic Test Case

5.2.1 Overview

The deterministic test case is run with a domain size of 100 m in the x-direction and 200 m in the y-direction. The landscape resolution of the simulation is 30 m, whereas the perimeter and distance resolutions are both 1 m. The total time of the simulation is 540 seconds, with a fire front produced at every time step of 60 seconds. The fuel was input as dry, mixed grass with shrub. The wind speed was input as 4 m/s, rather than the observed 2.5 m/s, to account for the difficulty in characterizing the fuel since there was no data on spatialization of the vegetation mixture. Wind direction was input as 345^{o} southeast, as observed.

5.2.2 Results

The result of the deterministic FARSITE run without data assimilation shows good comparison between the predicted front to the observed fire front. For all 9 time steps, the free-run firelines appear to have a slightly faster ROS than the observed maximum ROS of 0.25 m/s. However, even though the simulated wind speed input was 4 m/s, which was faster than the observed average wind speed of 2.5 m/s, the simulated maximum ROS was 0.1 m/s. The disagreement of ROS could be attributed to the fuel characterization in the input file. In addition, the FARSITE model over-predicts the flank spread as compared with observation flank spread.

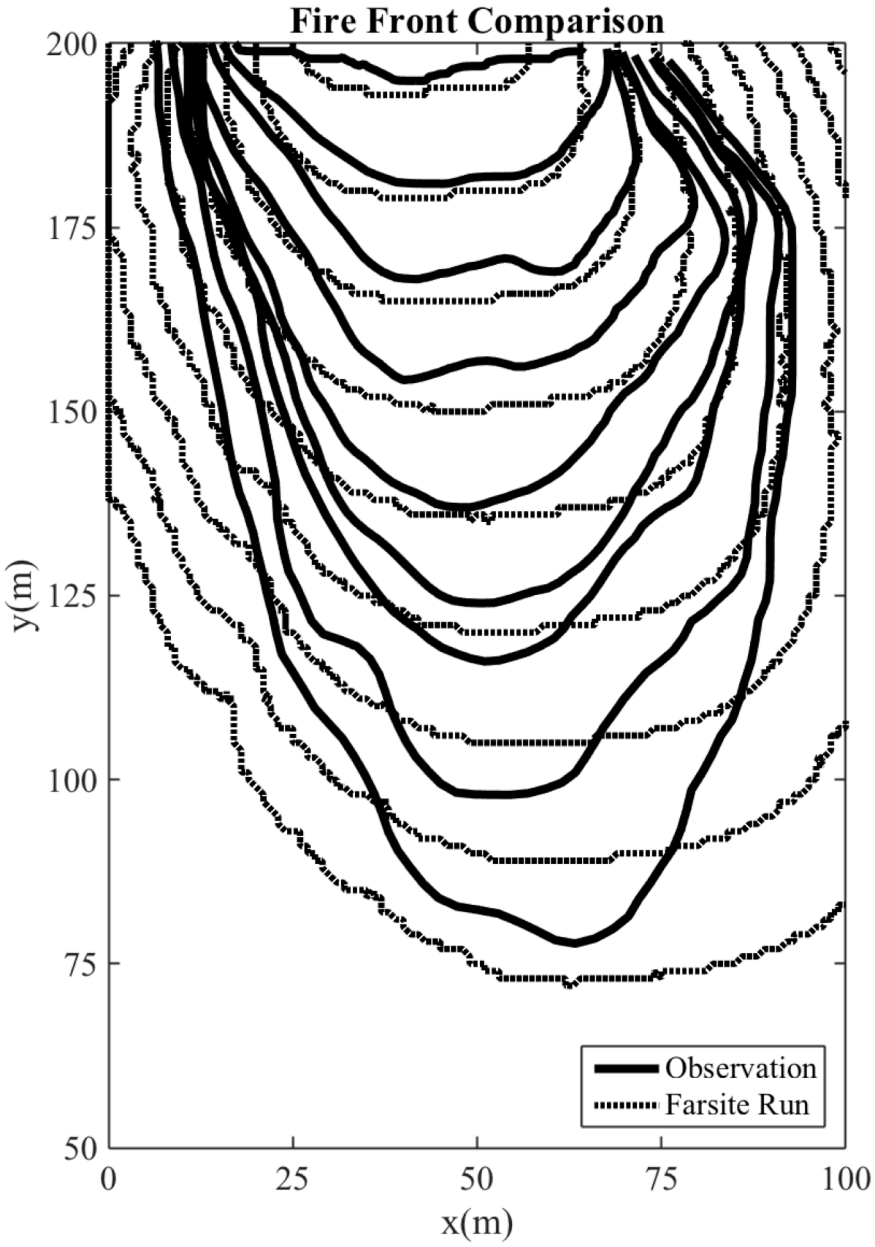

Figure 5.2 Deterministic run of the RxCADRE S5 experiment as simulated by FARSITE and compared to observation data. The time step between each front is 60 seconds.

5.2 Ensemble Test Case

5.2.1 Overview

The ensemble test case uses the same model simulation inputs as the deterministic test case. The domain is 100 m by 200 m, the landscape resolution is 30 m, and the distance and perimeter resolutions are 1 m. The simulation produces 60 ensemble members over 4 slave processors at each time step of 60 seconds, for a total simulation time of 540 seconds. The standard deviation for the mean 4 m/s wind speed is 3 m/s and the standard deviation for the mean 345^{o} wind direction is 10^{o}.

5.2.2 Results

The ensemble test case result shows that, similar to the FireFlux I ensemble test, the FARSITE-EnKF is capable of producing a predicted fireline that more closely matches the observed fireline. While the deterministic run already showed that the FARSITE model forecasted the fire front similar to observation, the EnKF will be able to correct and account for input errors in the simulation. Specifically, the difference in wind speed input and the difficulty in characterizing the heterogeneous, patchy fuel could attribute to the error.

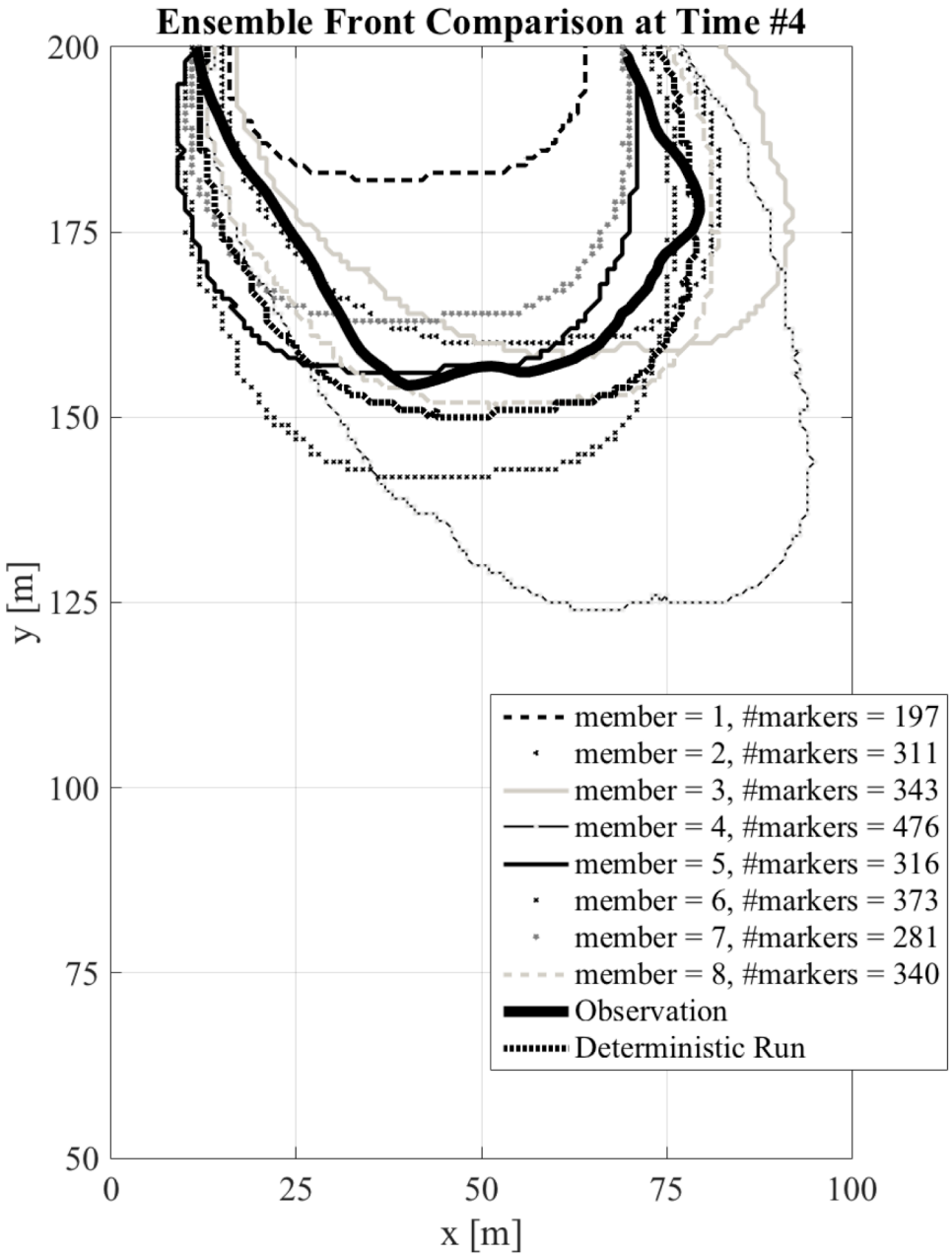

Figure 5.3 Comparison of 8 randomly selected ensemble members to the deterministic result and to the observed fireline data at time = 240 seconds.

Chapter 6: Current and Future Work

The preliminary FARSITE-EnKF model, without inverse capability, demonstrates an improvement in fireline prediction as compared with the standalone FARSITE model. In progress for this model now is the completion of the EnKF inverse loop integration within OpenPALM. The work requires Fortran and shell computer coding within the OpenPALM environment to allow communication between the forward model and the inverse algorithm. The error between the observation and predicted fireline is corrected in this step and the ensemble is assimilated to recursively produce a best-estimate parameter control vector and corresponding forecast.

After the FARSITE-EnKF model is completely integrated and the inverse loop is running properly, wind spatialization should be accounted for within the forward model. WindNinja, developed by Jason Forthofer et al. in 2007, is a computer application that is used within Windows FARSITE to allow for computation of spatially varying wind fields, however, it is not yet integrated with the FARSITE DLL. Another consideration for wind input is allowing the model to use non-integer wind speeds. As of now, when an ensemble is produced with FARSITE-EnKF, the input members do not recognize any decimal wind speeds and automatically correct themselves to input the wind to the nearest integer velocity. Using wind speeds to the 10^{th} decimal place would allow for greater input variation in the ensemble and, therefore, a wider array of state estimation outputs.

Another way to increase state estimation outputs is to simply increase the number of input ensemble members. For the purpose of saving computer cost and quickly running simulations, this project demonstrated FARSITE-EnKF forecast capability with 60 ensemble members. More ensemble members require more CPUs and a longer simulation running time, but would produce more forecasted firelines at each time step. In addition, the distribution of probability for the control parameters would more closely align with a Gaussian distribution, increasing the chances for a more accurate state estimation.

After validation of the FARSITE-EnKF model using wind speed and wind direction in the parameter control vector, other variables may be explored for model validation. Moisture content, vegetation type, and fuel bed depth are among other parameters that may be spatialized and perturbed at each time step within the simulation. This type of model validation would require the use of data for a larger fire with more independent variables.

Chapter 7: Conclusion

The motivation for this research is the need for an operational wildfire forecast model that produces data faster than real-time to improve fire management and fire suppression techniques. With such a tool, the threat of wildfires taking lives, destroying property, and releasing harmful emissions is drastically reduced. However, modeling wildfires is a difficult task, as the model must incorporate fire dynamics, weather data, topography, fuel characterization, and atmospheric interactions. There are few existing research-level models that couple wildfire-atmospheric forecast capabilities, but the data assimilation scheme in these models is computationally expensive and the outputs are at a low resolution. Data assimilation is a validated technique for other geophysical modeling and the methodology needs to be improved for wildfire models.

There are many ways to perform data assimilation for wildfire models. The FARSITE-EnKF method presented in this work is an improvement to the existing models in a few ways. FARSITE, being the most widely used operational wildfire model, simulates flank-spread of the fire front better than previous coupled programs. In addition, the model has been validated against many past fires, which supports that it provides a robust backbone for the physical description of fire dynamics (Finney, 2000). The EnKF, which relies on a stochastic description of the model behavior, is the selected data assimilation technique because it uses polynomial representation of the FARSITE forward model to the varying input parameters. This reduces computer cost and provides large sample of realizations, or ensemble members, while also

characterizing the model uncertainties. With a description of error provided in the model, the user can understand how accurate the simulated forecast is.

OpenPALM is the chosen dynamic coupling program for FARSITE-EnKF because it provides an environment that is pre-loaded with parallel computing capabilities. It allows for data exchange, intermediate computations, grid-to-grid interpolation, and parallel data redistribution. The algorithm of FARSITE-EnKF in OpenPALM is constructed with a combination of application-provided and user-written code. The final steps in the implementation of the inverse model is still pending. However, the statistical ensemble model is completed and provides results that support this research with a demonstration using FireFlux I data.

FireFlux I is a comprehensive, field-scale experiment that has been used to validate other wildfire-atmospheric coupled models. It is also used in this work to examine the forecast capabilities of the coupled FARSITE-EnKF model. Although it was conducted on a flat, homogeneous, grassland plot, eliminating independent variables such as vegetation type and topography, the experiment allows for a wind-focused model comparison. Therefore, only wind speed and wind direction are the parameters studied in the control vector because they most affect the position of the fire front. Deterministic test results show that FARSITE has the capability to spread head fire and flank fire, but it does not produce firelines that are entirely accurate with observed data. The ensemble test case results show that a 60-member stochastic output of the state at each time step produces forecast firelines that align closely with

observation. This is a critical foundation for validation of the FARSITE-EnKF. When the inverse loop is completed, the 60 ensemble members will be recursively assimilated to 1 best-fit state output. In addition, the required statistical model of probability density is demonstrated to be functioning correctly according to the Gaussian distribution of outputs.

Another model demonstration is given in this project using the RxCADRE S5 experiment. This experiment is a smaller scale than FireFlux I, but shows that FARSITE has good forecast capability when the simulation inputs are perturbed. The EnKF, when completed, will be able to account for input error correction and find best-fit prediction using normal probability distribution of the control parameters at each time step.

Current and future work on the FARSITE-EnKF software includes the following objectives: completing the inverse model algorithm; validating the model for a large-scale fire; spatializing the wind field to account for variations in the x and y directions; parameterizing other inputs such as fuel moisture, fuel depth, fuel density, etc.; and parallelization with an atmospheric model to improve the understanding of wildfire-atmosphere interactions.

Appendices

Appendix A: MATLAB Scripts

A.1: Wind Input Adjustments

The following three scripts serve the purpose of automatically adjusting the wind speed and wind direction inputs in the FARSITE input file "name.input". The first script, Read Custom Input, uses the existing input file and reads the matrix that contains the wind data directly into MATLAB. The second script, Write Custom Input, copies the input file into a new file, but replaces the wind data with the data provided by the user. The third script, FARSITE Input, is where the user defines what inputs to vary and by what magnitude in Write Custom Input.

1. Read Custom Input

```
function Raws = ReadCustInput(InputFilename)

[fin,errmsg] = fopen(InputFilename,'rt');  %Open the file, in read-
only text mode.
if(fin < 0)  %Make sure the file was actually opened.  Otherwise,
error.
    error(errmsg);end;

%Read up to the target matrix

CurLine = fgetl(fin); %Read the first line
while(ischar(CurLine)) %Make sure a line was read
    if((length(CurLine) > 5) && (strcmp(CurLine(1:5),'RAWS:')))
%Look for the line beginning with "RAWS:"
        break;end; %If found, stop looping
    CurLine = fgetl(fin); %If not found, read the next line and
repeat.
end;
```

```
if(~ischar(CurLine)) %If the end of the file was hit, fgetl return a
non-string.
    fclose(fin);
    error('RAWS section not found.');
end;

NumRaws = str2double(strtrim(CurLine(6:end)));  %Use the remainder
of the RAWS: line to determine how many rows to scan.

Raws = textscan(fin,'%u %u %u %u %u %u %f %u %u %u',NumRaws); %Read
in the array.
%Each of "NumRaws" rows is read formatted according to the funky
looking string.
%The string says read 10 columns, all of them unsigned integer
numbers
%except column 7, which has a decimal place (fixed point notation)

fclose(fin);  %Close the file - we've read what we needed.

for i = 1:10
    Raws{i} = double(Raws{i}); %Convert all the integers to floating
point numbers - matlab doesn't like integers
end

Raws = [Raws{:}];%Group the answer into a single matrix.
```

2. Write Custom Input

```
function WriteCustInput(InputFilename, OutputFilename, RawsMatrix)

if(size(RawsMatrix,2) ~= 10)  %Make sure we have 10 columns given to
us.  (We'll check rows later)
    error(['Provided RAWS matrix is the wrong size!  Expected 10
columns, given ', num2str(size(RawsMatrix,2))]);
end

[fin,errmsg] = fopen(InputFilename,'rt');  %Open the file, in read-
only text mode.
if(fin < 0)  %Make sure the file was actually opened.  Otherwise,
error.
    error(errmsg);end;
[fout,errmsg] = fopen(OutputFilename,'wt');  %Open the file for
writing in text mode.
if(fout < 0)  %Make sure the file was actually opened.  Otherwise,
error.
    fclose(fin);
    error(errmsg);
end;

%Read up to the target matrix

CurLine = fgets(fin); %Read the first line, keeping the carriage
```

```
return at the end.
while(ischar(CurLine)) %Make sure a line was read
    fwrite(fout,CurLine); %Write the line to the output.
    if((length(CurLine) > 5) && (strcmp(CurLine(1:5),'RAWS:')))
%Look for the line beginning with "RAWS:"
        break;end; %If found, stop looping
    CurLine = fgets(fin); %If not found, read the next line and
repeat.
end;
if(~ischar(CurLine)) %If the end of the file was hit, fgetl return a
non-string.
    fclose(fin);
    fclose(fout);
    error('RAWS section not found.');
end;

NumRaws = str2double(strtrim(CurLine(6:end)));  %Use the remainder
of the RAWS: line to determine how many rows to scan.

if(NumRaws ~= size(RawsMatrix,1))
    fclose(fin);
    fclose(fout);
    error(['Provided RAWS matrix is the wrong size!  Expected ',
num2str(NumRaws), ' rows, given ', num2str(size(RawsMatrix,1))]);
end

for(i = 1:NumRaws)
    fgets(fin); %Read an input line, and discard it.
    fprintf(fout,'%u %u %u %04u %u %u %.2f %u %u %u
\n',RawsMatrix(i,:)); %Write out the array
end
%Each of "NumRaws" rows is written formatted according to the funky
looking string.
%The string says write 10 columns, all of them unsigned integer
numbers
%except column 7, which has a decimal place (fixed point notation)
with two
%significant figures after the decimal

CurLine = fgets(fin); %Read the next line, keeping the carriage
return at the end.
while(ischar(CurLine)) %Make sure a line was read
    fwrite(fout,CurLine); %Write the line to the output.
    CurLine = fgets(fin); %If not found, read the next line and
repeat.
end;

fclose(fin);  %Close the file - we've read it all.
fclose(fout);  %Close the file - we've written it all.
```

3. FARSITE Input

```
close all;
clear;
clc;
format short;

%- SIMULATION PATH NAME ------------------------------------------
-------
fin = './fireflux_in/fireflux.input';
fout = './fireflux_in/fireflux.input';
% Read original wind data
Raws = ReadCustInput(fin);
Raws(:,8)= ((Raws(:,8))+4); %wind speed column
Raws(:,9)= ((Raws(:,9))+25); %wind direction column

% Output new wind data
WriteCustInput(fin,fout,Raws);
```

A.2: Post Processing FARSITE Data

The following script, Compare Front Ensemble, is for use with the folder "OUT_ENSEMBLE", which contains 420 files of the 60 ensemble member outputs at each of the 6 time steps. The outputs in these files have been extracted in a delineated form to represent X and Y coordinates for ease in MATLAB plotting.

1. Compare Front Ensemble

```
clear all;
close all;
clc;
format long;
ind_fig = 0;


% PARAMETER SETTINGS ------------------------------------------------
-------
%%% pathname
pathname = './OUT_ENSEMBLE/';

%%% time of interest
time = 2;

%%% number of ensemble members
ne = 60;

%%% plot parameters
%%%%% computational domain reframe (-pfr)
pfr = 20000;
%%%%% computational domain boundaries [m]
xmin = 0;
xmax = 380;
ymin = 0;
ymax = 790;
%%%%% mesh step size [m]
dx_plot = 76;
dy_plot = 79;
%%%%% final plot frame
size_plot = 30;


% PLOT FRONT DATA ---------------------------------------------------
-------

%%% colormap
cmap = colormap(jet(ne));
```

```
%%% figure plot: yfr = f(xfr) at a given time
ind_fig = ind_fig + 1;
f = figure(ind_fig);
axesf = axes('Parent',f,...
    'FontName', 'Times',...
    'FontSize',16,...
    'Xtick',[xmin:dx_plot:xmax],...
    'Ytick',[ymin:dy_plot:ymax],...
    'DataAspectRatio',[1 1
1]);set(f,'PaperUnits','centimeters','PaperPosition',[0 0 30 20])
hold on;
box on;
grid on;
axis([xmin xmax ymin ymax])
title(sprintf('10 Member Front Comparison at Time
#%d',time),'FontName', 'Times','FontSize',16);
xlabel('x [m]','FontName', 'Times', 'FontSize',16);
ylabel('y [m]','FontName', 'Times', 'FontSize',16);




%%% loop over ensemble of members
for ik = 1:ne

    %%%%% access to the front data for a given time indexed by 'it'
    filename = [pathname
sprintf('farsite_front_e%0.5d_MODspace_t=%0.5d.out',ik,time)];
    d = importdata(filename);
    xfr = d(:,1)-pfr;
    yfr = d(:,2)-pfr;
    nfr = length(d);

    %%%%% plot over 2D horizontal plane
    plot([xfr; xfr(:,1)],[yfr; yfr(:,1)],'LineStyle','-
','LineWidth',2,'Color',cmap(ik,:),'Displayname',...
        sprintf('member = %d, #markers = %d',ik,nfr));
    hold on;
end

%%%Filippi's observation front
 tcurr = 120;
    filename = [sprintf(['./filippi_front/fireflux_t',
sprintf('%0.3d',tcurr) '.txt'])];
    cfd_fronts = importdata(filename);
    b = cfd_fronts(:,[3,4]);
    N_dis = 1;
    N_total = length(b);
    h2 = plot(b(1:N_dis:N_total,1)-380,b(1:N_dis:N_total,2)-
760,'ko','MarkerSize',4,'MarkerFaceColor','k');
    hold on;



%%% add legend on front data plot
%legend(axesf,'show','Location','SouthEastOutside');
```

```
colorbar('peer',axesf,'FontSize',16);

%%% plot settings
set(gcf,'PaperPosition', [0 0 size_plot size_plot]);
set(gcf,'PaperSize', [size_plot size_plot]);
```

Appendix B: Application Installation Procedures

B.1: Procedure to Install OpenPALM within DeepThought2

The following are full, detailed instructions for installing OpenPALM on DT2, written by Melanie Rochoux.

PART 1: Accessing DeepThought2

1. Obtaining permission to access DeepThought2:

Professor Arnaud Trouve can email the DeepThought2 administrators in order to get an account approved for you if you activate your TerpConnect account and provide him with your DirectoryID.

2. Download and install an SSH client:

All access to DeepThought2 is done remotely using the Secure Shell (SSH) protocol. As such, you need to obtain an SSH client. PuTTY is the most common SSH client and can be obtained here, but I prefer to use MobaXterm because it simplifies the X11 forwarding process. Access to DeepThought2 is restricted to machines on the umd.edu domain; therefore, if one wishes to access DeepThought2 from outside the UMD campus, one needs to access UMD's virtual private network (VPN) first. UMD's VPN client can be obtained through TerpWare.

3. Accessing DeepThought2:

At this point, logging into the DeepThought2 cluster requires proper configuration of the SSH client. Use login.deepthought2.umd.edu (Port 22) as the remote host name and your TerpConnect credentials.

NOTE: Online documentation on DeepThought2 is available here: http://www.glue.umd.edu/hpcc/help/usage.html

PART 2: Installing OpenPALM

This document is designed to be used alongside documentation from CERFACS found here: http://www.cerfacs.fr/globc/PALM_WEB/EN/DOCUMENTS/manuals.html

1. Create a directory to contain the OpenPALM files:

```
mkdir $HOME/PalmFiles
```

2. Obtain the OpenPALM files:Download a compressed .tar archive containing all the OpenPALM files from the CERFACS website: http://www.cerfacs.fr/globc/PALM_WEB/EN/BECOMEAUSER/instructions.html

3. Put the distrib.tgz file into the $HOME/PalmFiles directory. (This might require a FTP client).

4. Uncompress the archive with this command:

```
tar –xvf distrib.tgz
```

5. Setup environment variablesIn order for OpenPALM to function properly, it must be told the paths to various softwares on DeepThought2. These include the location of the MPI files and compilers for FORTRAN, C and C++. Because it is very easy to mistype the path to a particular directory, it is preferable to set environmental variables instead in the .cshrc.mine file in the home directory. Add the following lines in the .cshrc.mine file (csh):

```
setenv PALM_DIR $HOME/FIREFLY/PalmFiles
setenv PALM_MP $PALM_DIR/PALM_MP
setenv PALMHOME $PALM_MP/linux64r4_intel
setenv PREPALMMPDIR $PALM_DIR/PrePALM_MP
alias prepalm '$PREPALMMPDIR/prepalm_MP.tcl \!* &'
```

Note that the recommended installation on DeepThought2 relies on Intel compilers. There are no PGI (Portland Group) compiler licences availible and GNU compilers have not been tested yet.

6. Install the STEPLANG interpreterInstructions are found in chapter 3.3 of the CERFACS' linux installation guide. Because Intel compilers are used, make sure to modify the makefile so that CC = icc.

7. Installing OpenPALMIn order to install OpenPALM, the Makefile file must be appropriately configured. Use these commands:

```
 cd $HOME/PalmFiles/PALM_MP

./configure --prefix=/homes/porterw/PalmFiles/PALM_MP/linux64r4_intel --with-
CC=mpiicc –
with-F90=mpiifort --with-
openmpi=/cell_root/software/intel/ics_2013.1.039/impi/4.1.1.036/intel64/ --enable-
64bits --with- shared_lib LDFLAGS=-Wl,-
```

```
rpath,/cell_root/software/intel/ics_2013.1.039/composer_xe_2013_sp1.0.080/compile
r/lib/intel64

make clean
make
make install
```

NOTE: The configure command that successfully installed OpenPALM on DT2 is written in the file "config.log.success" in the following directory: homes/porterw/PalmFiles/PALM_MP/config.log.success

8. Verifying installation and set compilation optionsVerify that the installation is done properly by completing the first five tutorials in CERFACS' user guide. For all the tutorials, use the following options (replace PALMHOME with your own home directory) in the Make.include file:

```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ #
PALMHOME = /homes/porterw/PalmFiles/PALM_MP/linux64r4_intel/

F90 = mpiifort
F90FLAGS = L
F90FLAGS =
F90EXTLIB =

F77 = mpiifort
F77FLAGS = L
F77FLAGS =
F77EXLIB =

FPPFLAGS = -cpp

CC = mpiicc
CCFLAGS = -std=c99
LCCFLAGS =
CCEXTLIB = -lmpi_f90 -lmpi_f77

C++ = mpiicpc
C++FLAGS = -DMPICH_IGNORE_CXX_SEEK
LC++FLAGS =
```

```
C++EXTLIB = -lifport -lifcpre

OMPFLAGS =

SOFLAGS = -shared -fpic

INCLUDES = -I/usr/local/include
LIBS =

PYTHON = python
CYTHON = cython
PYTHON_INCLUDE = `${PYTHON} -c 'from distutils import sysconfig; print(
sysconfig.get_python_inc() )'` MPI4PY_INCLUDE = `${PYTHON} -c 'import
mpi4py; print( mpi4py.get_include() )'`
NUMPY_INCLUDE = `${PYTHON} -c 'import numpy; print( numpy.get_include()
)'`

USERINCF =
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ #
```

9. Running OpenPALM jobs on DeepThought2Since DeepThought2 is a shared supercomputer with a queue system, the procedure to run jobs is important to follow and running the OpenPALM driver through the stand-alone PALM commands must be avoided. The following procedure is recommended:

a) Generate PALM service files, either by loading the PrePALM graphical user interface with the command prepalm or directly with the command:

```
$PREPALMMPDIR/prepalm_tclsh.tcl –no-make-include –c *.ppl
```

*where * must be replaced by the name of the PrePALM file (whose extension is ppl). Note that the option –no-make-include avoids to erase the Make.include file already built with the correct compilation options. Note also that the prepalm command is recommended when the application is run from a remote station.*

b) Generate the executable program of the OpenPALM driver:

```
make
```

c) Run the OpenPALM driver:

- *Use the mpdboot command instead of lamboot to launch the MPI process (mpd &).*
- *Run the following script run.job using* `sh run.job` .

```
sbatch palm.script
squeue -A firemodel-hi
sbalance -all --nosuppress0
```

This script runs the script palm.script that defines the running options (computational time, number of cores, run directory, etc.). More informations can be found on the DeepThought2 website (http://www.glue.umd.edu/hpcc/help/jobs.html). Here is a simple example:

```
#!/bin/tcsh
#SBATCH -A firemodel-hi
#SBATCH -t 00:15:00
#SBATCH --ntasks=1
module load intel
set WORKDIR=/homes/....
cd $WORKDIR
pwd

mpirun -wdir $WORKDIR ./palm_main
```

B.2: Procedure to Install FARSITE within OpenPALM

Once the setup of OpenPALM is complete and all prerequisite libraries installed, FARSITE can be implemented and simulations may be commanded from the DT2 terminal. The following are full, detailed instructions for integrating and running FARSITE with OpenPALM, written by Cong Zhang and Melanie Rochoux.

1. Create a directory to contain the FARSITE run directory:

```
mkdir $HOME/FARSITE
```

2. Obtain the FARSITE run directory and copy it into "$HOME/FARSITE".

Send an email to: cong0129@umd.edu for the latest version.

3. Install the shapefile library:

Download the shapefile libarary from http://download.osgeo.org/shapelib/. Create a new directory (/homes/cong0129/FireLib/) to contain this library, follow README instruction to build it. You may need to set lib path in the user profile ".cshrc.mine" file.

```
setenv PATH $PATH\:/homes/cong0129/FireLib/shapelib-1.3.0/bin
```

4. Setup makefile options:

Use the same Make.include file as for the FIREFLY training sessions:

```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ #
```

```
PALMHOME = /homes/cong0129/PalmFiles/PALM_MP/linux64r4_intel

F90 = mpiifort
F90FLAGS =
LF90FLAGS = -lstdc++
F90EXTLIB = -lstdc++

F77 = mpiifort
F77FLAGS =
LF77FLAGS =
F77EXLIB =

FPPFLAGS = -cpp

CC = mpiicc
CCFLAGS = -std=c99
LCCFLAGS =
CCEXTLIB = -lmpi_f90 -lmpi_f77

C++ = mpiicpc
C++FLAGS = -DMPICH_IGNORE_CXX_SEEK
LC++FLAGS =
C++EXTLIB = -lifport -lifcpre

OMPFLAGS =

SOFLAGS = -shared -fpic

INCLUDES = -I/usr/local/include
LIBS = /homes/cong0129/FireLib/lapack-3.5.0/*.a
/homes/cong0129/FireLib/BLAS/blas_LINUX.a

PYTHON = python
CYTHON = cython
PYTHON_INCLUDE = `${PYTHON} -c 'from distutils import sysconfig; print(
sysconfig.get_python_inc() )'`
MPI4PY_INCLUDE = `${PYTHON} -c 'import mpi4py; print( mpi4py.get_include()
)'`
NUMPY_INCLUDE = `${PYTHON} -c 'import numpy; print( numpy.get_include()
)'`

USERINCF =

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ #
```

5. Compile FARSITE on DeepThought2:

a) Generate PALM service files, either by loading the PrePALM graphical user interface with the command "prepalm" or directly with the command:

```
$PREPALMMPDIR/prepalm_tclsh.tcl --c PLATFORM_FARSITE_E.ppl
```

Note that the "prepalm" command is recommended when the application is run from a remote station.

b) Generate the executable program of the OpenPALM driver:

```
make -f makefile_slaves_RUN_FARSITE_E
make
```

NOTE: When performing multiple compilations of the source code (due to code development for instance), it is important to clean the environment through the command make allclean *and to compile again the whole OpenPALM-based application: the compilation procedure described above (5a-b) must be performed again.*

6. Run the OpenPALM-based FARSITE application

- *Use the mpdboot command to run the MPI process (mpd &).*
- *Run the following script run.job* sh run.job.

```
sbatch Farsite.script
squeue -A firemodel-hi
sbalance -all --nosuppress0
```

This script runs the script Farsite.script that defines the running options (computational time, number of cores, run directory, etc.) such as:

```
#!/bin/tcsh
#SBATCH -A firemodel-hi
#SBATCH -t 00:60:00
#SBATCH --ntasks=20
#SBATCH --ntasks-per-core=1

module unload openmpi
module load intel/2013.1.039
limit stacksize unlimited

date
mpirun -np 1 ./palm_main
date
```

NOTE: The additional option "limit stacksize unlimited" *is important to increase the stack memory*.

Bibliography

Andrews, P L. "Modeling Wind Adjustment Factor and Midflame Wind Speed for Rothermel's Surface Fire Spread Model." *USDA Forest Service - General Technical Report RMRS-GTR* 266 (2012): 1–39. Web.

Buehner, Mark et al. "Intercomparison of Variational Data Assimilation and the Ensemble Kalman Filter for Global Deterministic NWP. Part II: One-Month Experiments with Real Observations." *Monthly Weather Review* 138.5 (2010): 1567–1586. Web.

Carrassi, Alberto, and Stéphane Vannitsem. "Accounting for Model Error in Variational Data Assimilation: A Deterministic Formulation." *Monthly Weather Review* 138.9 (2010): 3369–3386. Web.

Clements, Craig B. et al. "Observing the Dynamics of Wildland Grass Fires: FireFlux - A Field Validation Experiment." *Bulletin of the American Meteorological Society* 88.9 (2007): 1369–1382. Web.

Coen, Janice L. et al. "Wrf-Fire: Coupled Weather-Wildland Fire Modeling with the Weather Research and Forecasting Model." *Journal of Applied Meteorology and Climatology* 52.1 (2013): 16–38. Web.

Delmotte, Blaise, Charlotte Emery, and Roberto Paoli. "A Regional-Scale Data-Driven Wildfire Spread Simulator." 2 (2014): n. pag. Print.

Descombes, G. et al. "Generalized Background Error Covariance Matrix Model (GEN_BE v2.0)." *Geoscientific Model Development* 8.2004 (2015): 669–696. Web.

Durand, Mickael. "Data-Driven Wildfire Spread Modeling." 1319 (2015): n. pag. Print.

Filippi, Jean-baptiste, Xavier Pialat, and Craig B Clements. "Assessment of FOREFIRE / MESONH for Wildland Fire / Atmosphere Coupled Simulation of the FireFlux Experiment." (2014) 1–18. Print.

Finney, Mark A. "Efforts at Comparing Simulated and Observed Fire Growth Patterns." (2000): 20. Print.

Finney, Mark a, Dai Qin Ã, et al. "FARSITE : Fire Area Simulator — Model Development and Evaluation." *Evaluation* Research P.February (1998): 47. Web.

Finney, Mark a, Jack D Cohen, et al. “On the Need for a Theory of Wildland Fire Spread.” *International Journal of Wildland Fire* 22.1 (2013): 25–36. Web.

Gall, Robert, David Mccarren, and Fred Toepfer. “Deterministic vs . Ensemble Forecasts : The Case from Sandy.” 3.2 (2013): 5–11. Print.

Hyde, Kevin et al. “Research and Development Supporting Risk-Based Wildfire Effects Prediction for Fuels and Fire Management: Status and Needs.” *International Journal of Wildland Fire* 22.1 (2013): 37–50. Web.

Kochanski, A. K. et al. “Evaluation of WRF-SFIRE Performance with Field Observations from the FireFlux Experiment.” *Geoscientific Model Development* 6.4 (2013): 1109–1126. Web.

Linn, Rodman R, and Francis H Harlow. “Submitted to S SECOND SYMPOSIUM ON FIRE AND FOREST.” 836.1 046 (1997): n. pag. Print.

Mandel, J et al. “Data Assimilation for Wildland Fires: Ensemble Kalman Filters in Coupled Atmosphere-Surface Models.” *Control Systems Magazine, IEEE* 29.3 (2009): 47–65. Web.

Mell, We, Rj McDermott, and Gp Forney. “Wildland Fire Behavior Modeling: Perspectives, New Approaches and Applications.” *Firescience.Gov* (2010): 4. Web.

Papadopoulos, George D, and Fotini-niovi Pavlidou. “A Comparative Review on Wild Fi Re Simulators.” 5.2 (2011): 233–243. Print.

Piacentini, Andrea. “High Performance Computing for Computational Science --- VECPAR 2002: 5th International Conference Porto, Portugal, June 26--28, 2002 Selected Papers and Invited Talks.” Ed. José M L M Palma et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. 479–492. Web.

Riverain, Marion, Valentine Chatel, and Céline Vargel. “Data Assimilation for Wildfire Spread Forecasting : Comparison of Rate of Spread Models.” February (2015): n. pag. Print.

Robinson, Allan R, and Pierre F J Lermusiaux. “Ocean Science by.” 62 (2000): 1–12. Print.

Rochoux, M. C. et al. “Towards Predictive Data-Driven Simulations of Wildfire Spread - Part I: Reduced-Cost Ensemble Kalman Filter Based on a Polynomial Chaos Surrogate Model for Parameter Estimation.” *Natural Hazards and Earth System Sciences* 14.11 (2014): 2951–2973. Web.

Rochoux, Mélanie C. et al. “Data Assimilation Applied to Combustion.” *Comptes Rendus - Mecanique* 2013: 266–276. Web.

Rothermel, Richard C. “A Mathematical Model for Predicting Fire Spread in Wildland Fuels.” *USDA Forest Service Research Paper INT USA* INT-115 (1972): 40. Web.

Sakov, Pavel, Geir Evensen, and Laurent Bertino. “Asynchronous Data Assimilation with the EnKF.” *Tellus, Series A: Dynamic Meteorology and Oceanography* 62 (2010): 24–29. Web.

Scott, Joe H. “Introduction to Fire Behavior Modeling.” (2012): 0–149. Print.

Tymstra, C et al. *Development and Structure of Prometheus : The Canadian Wildland Fire Growth Simulation Model*. N.p., 2010. Web.

USGS. “LANDFIRE.” Accessed 2014 (2014): n. pag. Web.

Welch, Greg, and Gary Bishop. “An Introduction to the Kalman Filter.” Ed. Addison-WesleyEditor Acm Press. *In Practice* 7.1 (2006): 1–16. Web.

Wells, Gail. “The Rothermel Fire-Spread Model: Still Running Like a Champ.” *Fire Science Digest* 2 (2008): 1–12. Web.

Wells, Gail. “RxCADRE Takes Fire Measurements to Whole New Level.” 16 (2013): n. pag. Web.

“WindNinja Select Publications & Products.” (2014): n. pag. Web.

WMO Weather, World. “Verification Methods for Tropical Cyclone Forecasts.” November (2013): n. pag. Print.