Fire Dynamics Repository Manual

M. Currie
mileshcurrie@gmail.com

July 27, 2017

1 Preface

This manual is meant to provide a quick introduction to the "FireDynamics" repository I have created on GitHub. This is by no means a step-by-step quide on how to run the scripts, however it will focus on the big picture ideas of how things run. I have made it a point to document all of the scripts very well, so the hope is that with this manual and the documentation in the code, a user should be able to run everything with little instruction. There will be some details that I will unintentionally leave out, so I invite the user to contact me with any questions if they feel that a certain section is not intuitive enough.

2 firePropagationModel.py

2.1 Description

This script models the propagation of a fire based on heuristic rules that were deduced by observation. The driving factor in the propagation is the probability of a cell that is on fire to light one of its neighbors on fire. If one inspects the functions that have to do with probability, the made-up heuristic rules can be seen clearly. These rules are always changing, so this document will be edited with more detail later.

The idea for this model is to start out with some initial fire, which can be specified by the user to be a point, line, parabola, or the initial frame from a real dataset and use the heuristic probability rules to let the fire propagate for the specified number of timesteps. In addition to the parameter for the number of timesteps, the other parameters that can be specified are how big the user wants the propagation frame to be (N), how far the fire can spread in one timestep (numLayers), the number of timesteps that a cell can stay on fire for (masterBurnTime), and two flags that specify whether the user wants to plot and save the resulting images.

The basic structure of this script is as follows: The script loops over the number of specified timesteps and for each timestep, it calculates a new frame in which the fire has propagated. For each iteration, the script looks for pixels that are on fire, which are flagged by a positive integer. For each of the on fire pixels, the script "rolls a dice" to determine if its neighbors light on fire in the next frame. The probabilities are based on the distance from the pixel of interest and some master probability that can be spceified by the user. If some on fire pixel reaches the number specified with the masterBurnTime parameter, the pixel is flagged as a "-1", which means that pixel is burnt and can no longer light on fire.

After the script has run through its iterations, it plots and saves the images (if these flags are set to "True"), which shows the propagation.

2.2 Companion Scripts

There are two companion scripts to this one: makeImages.py and makeMovies.py. The functions of these scripts are just as intuitive as they sound. The former takes in the path to the numpy files that were generated from the model and creates/saves images of them. The keyword arguments in the creation of

the images have been specifically tuned for fires to make them both visually appealing and consistent. The latter script takes the images generated and uses ffmpeg to create an mp4 movie so the user can see the fire propagate.

3 DMD_reconstruction.py

This script was originally created by Alessandro Alla to perform a dynamic mode decomposition (DMD) analysis on the real data that is available. The script was originally written in MATLAB, but I translated it to python to make it run better on my machine. Presently, it is unclear what we are going to accomplish using this type of analysis, therefore, I won't elaborate much on the subject. If the user is interested enough, I would suggest reading up on DMD to see some of its capabilities.

4 ThermalCubeAnalysis.py

Again, this script was not written by me. It was aquired from Kevin Speer to analyze the data from the fire box experiment that we have occasionally conducted. This is a simple script that takes in a csv file and plots the temperature and heat flux from the fire. In theory, once the gas sensors on the box are up and running, this script will also plot statistics for the humidity in the air and the oxygen levels in the air. At the time of writing, the latter features were not up and running on the fire box.

5 interpretDMD.py

This script is basically an amalgamation of the scripts "makeImages.py" and "makeMovies.py" but tailored to the results of the DMD reconstruction code.

6 aggregatePixels.py

This script takes in a three dimensional numpy file containing temperature data for a fire and aggregates pixels together in square blocks. For example, if the user sets the aggregation level to three for a dataset, the script will group together blocks of dimension three by three pixels, take the maximum value of that group, and assign it to a new cell in a new dataset. This effectively makes the resolution worse, which in turn makes the subsequent scripts run faster and may treat spotting effects better.

7 computeDiffCoeff.py

This script is purely experimental and should not be assumed to produce reliable results. At the time of writing, this was mostly abandoned as an initial attempt at calculating the diffusion coefficients for a given timeseries dataset. I will not elaborate further as this is not actively being worked on.

8 fireStats.py

This script is the main analysis script that has been under development during the summer of 2017. This script performs several analyses: It gives some basic statistics for a given dataset, calculates the burn times for each pixel in the dataset, calculates the probability of a cell lighting on fire, and counts the number of cells surrounding a pixel of interest that are on fire.

8.1 Basic Statistics

This section of the script produces plots of the most basic statistics for a dataset. For each frame, the script calculates the mean, median, minimum, maximum, and standard deviation temperature values. After this has been calculated for all of the frames, the script plots all of these values, with frame number on the x-axis and statistic (temperature) value on the y-axis. If the user wants to run this section of the script, they should set the "plotTempStats" flag to "True".

8.2 Burn Times

This section of the script calculates how long each pixel stays on fire. Each pixel carries with it an integer value that signifies how many timesteps that pixel has been on fire for at the end of the timeseries. The script then plots these values as a heat map: redder colors signify pixels that stay on fire for longer and bluer pixels signify pixels that had relatively short burn times. If the user wants to run this section of the script, they should set the "plotBurnTimes" flag to "True".

8.3 Counts

This section of the script loops over all of the pixels in a frame in a certain timestep and looks in the immediate vicinity of a pixel for other pixels that were on fire in the previous timestep that might have lit the pixel of interest on fire. This counts up the surrounding burning pixels and separates the two cases of whether or not the pixel of interest lit on fire because of these pixels. In the end, there will be two lists: one which counts the number of neighboring pixels that DID light the pixel of interest on fire and one which counts the number of neighboring pixels that DID NOT light the pixel of interest on fire. The script then bins these lists by count and displays a histogram. If the user wants to run this section of the script, they should set the "plotHistsAndProbs" to "True".

8.4 Combustion Probabilities

This section of the script takes the lists of counts mentioned in section 8.3 and uses them to calculate probabilities by the formula:

If the user wants to run this section of the script, they should set the "plotHistsAndProbs" to "True".

9 Complete_Video_Analysis.py

This script was compiled and created by Elizabeth Tilly and Scott Goodrick. This program is a comprehensive video analysis script designed specifically for analyzing smoke and fire fluid dynamics. Included are particle imaging velocimetry, principal component analysis, and graphing programs. See the script for more documentation. My recommended running example:

```
import Complete_Video_Analysis as cva

cva.pivprocess('/Path/To/MP4',1,3500,199.25,24.,'/Path/To/Save/Directory/%05d.npz',

 '/Path/To/Save/Directory/%05d.png',startframe=0, contouralpha=0.65,

vertvelmin=-8,vertvelmax=8,hozvelmin=0,hozvelmax=6)
```

This will produce velocity vectors for each frame in a dataset.