The GridFire Fire Behavior Model

Gary W. Johnson, Ph.D., David Saah, Ph.D., Max Moritz, Ph.D.

1 Preface

This document is a Literate Program[1], containing both the source code of the software it describes as well as the rationale used in each step of its design and implementation. The purpose of this approach is to enable anyone sufficiently experienced in programming to easily retrace the author's footsteps as they read through the text and code. By the time they have reached the end of this document, the reader should have just as strong a grasp of the system as the original programmer.

To execute the code illustrated within this document, you will need to install several pieces of software, all of which are open source and/or freely available for all major operating systems. These programs are listed in Table 1 along with their minimum required versions and URLs from which they may be downloaded.

Table 1: Software necessary to evaluate the code in this document

Name	Version	URL
Java Development Kit	1.7+	http://www.java.com
Leiningen	2.5.2+	http://leiningen.org
Boot	2.1.2+	http://boot-clj.com
Postgresql	9.3+	http://www.postgresql.org
PostGIS	2.1+	http://postgis.net

GridFire is written in the Clojure programming language[2], which is a modern dialect of Lisp hosted on the Java Virtual Machine.(Hickey, 2008) As a result, a Java Development Kit is required to compile and run the code shown throughout this document.

The Clojure build tools, Leiningen and Boot, are used to download required libraries and provide a code evaluation prompt (a.k.a. REPL) into which we will enter the code making up this fire model.

[1] https://en.wikipedia.org/wiki/Literate_programming

[2] http://clojure.org

Postgresql (along with the PostGIS spatial extensions) will be used to load and serve raster-formatted GIS layers to the GridFire program. Although it is beyond the scope of this document, PostGIS provides a rich API for manipulating both raster and vector layers through SQL. See http://postgis.net for more information.

2 Setting Up the Clojure Environment with Leiningen

Because Clojure is implemented on the Java Virtual Machine (JVM), we must explicitly list all of the libraries used by our program on the Java classpath. Fortunately, the Clojure build program, Leiningen, can handle downloading and storing these libraries as well as linking them to the Clojure process at runtime. However, in order for Leiningen to know which libraries are needed, we must first create its config file, called "project.clj", and place it in the directory from which we will call our Clojure program. A minimal but complete project.clj is shown below.

```
(defproject sig-gis/gridfire "1.2.0"
  :description      "SIG's Raster-based Fire Behavior Model"
  :dependencies     [[org.clojure/clojure                "1.7.0"]
                     [org.clojure/data.csv               "0.1.3"]
                     [org.clojure/java.jdbc              "0.4.2"]
                     [postgresql/postgresql              "9.3-1102.jdbc41"]
                     [net.mikera/core.matrix             "0.42.0"]
                     [net.mikera/vectorz-clj             "0.36.0"]
                     [sig-gis/magellan                   "0.1.0"]
                     [org.clojars.lambdatronic/matrix-viz "0.1.7"]]
  :repositories     [["java.net"  "http://download.java.net/maven/2"]
                     ["osgeo.org" "http://download.osgeo.org/webdav/geotools/"]]
  :min-lein-version "2.5.2"
  :aot              [gridfire.cli]
  :main             gridfire.cli
  :repl-options     {:init-ns gridfire.cli}
  :global-vars      {*warn-on-reflection* true})
```

Once this file is created, we need to instruct Leiningen to download these library dependencies and then open the REPL (Read-Evaluate-Print-Loop).

```
lein deps
lein repl
```

3 Setting Up the Clojure Environment with Boot

As an alternative to Leiningen, the Clojure ecosystem provides another build tool called Boot. This program performs all of the same functions as Leiningen and is configured similarly by placing a text file in the directory from which the Clojure program will be run. In this case, two files are required: "boot.properties" and "build.boot".

```
BOOT_CLOJURE_NAME=org.clojure/clojure
BOOT_CLOJURE_VERSION=1.7.0
BOOT_VERSION=2.5.5
```

```
(task-options!
 pom  {:project     'sig-gis/gridfire
       :version     "1.2.0"
       :description "SIG's Raster-based Fire Behavior Model"}
 repl {:eval        '(set! *warn-on-reflection* true)
       :init-ns     'gridfire.cli}
 aot  {:namespace   '#{gridfire.cli}}
 jar  {:main        'gridfire.cli})

(set-env!
 :source-paths   #{"src" "test"}
 :resource-paths #{"resources"}
 :dependencies   '[[org.clojure/clojure                 "1.7.0"]
                   [org.clojure/data.csv                "0.1.3"]
                   [org.clojure/java.jdbc               "0.4.2"]
                   [postgresql/postgresql               "9.3-1102.jdbc41"]
                   [net.mikera/core.matrix              "0.42.0"]
                   [net.mikera/vectorz-clj              "0.36.0"]
                   [sig-gis/magellan                    "0.1.0"]
                   [org.clojars.lambdatronic/matrix-viz "0.1.7"]
                   [adzerk/boot-test                    "1.1.0" :scope "test"]]
 :repositories   #(conj %
                        ["java.net"  "http://download.java.net/maven/2"]
                        ["osgeo.org" "http://download.osgeo.org/webdav/geotools/"]))

(require '[adzerk.boot-test :refer :all])

(deftask build
  "Build my project."
  []
  (comp (aot) (pom) (uber) (jar)))

(deftask testing
  "Automatically run tests after each file save."
  []
  (comp (watch) (speak) (test)))
```

To download all the required dependencies and launch a REPL to follow along with the rest of this document, simply run the following command from your shell:

```
boot repl
```

4 Setting Up the PostGIS Database

GridFire may make use of any raster-formatted GIS layers that are loaded into a PostGIS database. Therefore, we must begin by creating a spatially-enabled database on our local Postgresql server.

When installing Postgresql, we should have been prompted to create an initial superuser called **postgres**, who has full permissions to create new databases and roles. We can log into the Postgresql server as this user with the following **psql** command.

```
psql -U postgres
```

Once logged in, we issue the following commands to create a new user account with our system login name (in my case, this is gjohnson). We then create a new database to store our raster data (e.g., gridfire) and import the PostGIS spatial extensions into it.

```sql
CREATE ROLE gjohnson WITH LOGIN CREATEDB;
CREATE DATABASE gridfire WITH OWNER gjohnson;
\c gridfire
CREATE EXTENSION postgis;
```

Whenever we want to add a new raster-formatted GIS layer to our database, we can simply issue the **raster2pgsql** command as follows, replacing the raster name and table name to match our own datasets.

```sh
SRID=4326
RASTER=dem.tif
TABLE=dem
DATABASE=gridfire
raster2pgsql -s $SRID $RASTER $TABLE | psql $DATABASE
```

Note: The raster2pgsql command has several useful command line options, including automatic tiling of the raster layer in the database, creating fast spatial indeces after import, or setting raster constraints on the newly created table. Run **raster2pgsql -?** from the command line for more details.

5 Fire Spread Model

GridFire implements the following fire behavior formulas from the fire science literature:

Surface Fire Spread: Rothermel 1972 with FIREMODS adjustments from Albini 1976

Crown Fire Initiation: Van Wagner 1977

Passive/Active Crown Fire Spread: Cruz 2005

Flame Length and Fire Line Intensity: Byram 1959

Midflame Wind Adjustment Factor: Albini & Baughman 1979 parameterized as in BehavePlus, FARSITE, FlamMap, FSPro, and FPA according to Andrews 2012

Fire Spread on a Raster Grid: Morais 2001 (method of adaptive timesteps and fractional distances)

The following fuel models are supported:

Anderson 13: no dynamic loading

Scott & Burgan 40: dynamic loading implemented according to Scott & Burgan 2005

The method used to translate linear fire spread rates to a 2-dimensional raster grid were originally developed by Marco Morais at UCSB as part of his HFire system.(Peterson et al., 2011, 2009, Morais, 2001) Detailed information about this software, including its source code and research article references can be found here:
http://firecenter.berkeley.edu/hfire/about.html
Outputs from GridFire include fire size (ac), fire line intensity (Btu/ft/s), flame length (ft), fire volume (ac*ft), fire shape (ac/ft) and conditional burn probability (times burned/fires initiated). Fire line intensity and flame length may both be exported as either average values per fire or as maps of the individual values per burned cell.

In the following sections, we describe the operation of this system in detail.

5.1 Fuel Model Definitions

All fires ignite and travel through some form of burnable fuel. Although the effects of wind and slope on the rate of fire spread can be quite pronounced, its fundamental thermodynamic characteristics are largely determined by the fuel type in which it is sustained. For wildfires, these fuels are predominantly herbaceous and woody vegetation (both alive and dead) as well as decomposing elements of dead vegetation, such as duff or leaf litter. To estimate the heat output and rate of spread of a fire burning through any of these fuels, we must determine those physical properties that affect heat absorption and release.

Of course, measuring these fuel properties for every kind of vegetation that may be burned in a wildfire is an intractable task. To cope with this, fuels are classified into categories called "fuel models" which share similar burning characteristics. Each fuel model is then assigned a set of representative values for each of the thermally relevant physical properties shown in Table 2.

Table 2: Physical properties assigned to each fuel model

Property	Description	Units
δ	fuel depth	ft
w_o	ovendry fuel loading	lb/ft^2
σ	fuel particle surface-area-to-volume ratio	ft^2/ft^3
M_x	moisture content of extinction	lb moisture/lb ovendry wood
h	fuel particle low heat content	Btu/lb
ρ_p	ovendry particle density	lb/ft^3
S_T	fuel particle total mineral content	lb minerals/lb ovendry wood
S_e	fuel particle effective mineral content	lb silica-free minerals/lb ovendry wood
M_f	fuel particle moisture content	lb moisture/lb ovendry wood

Note: While M_f is not, in fact, directly assigned to any of these fuel models, their definitions remain incomplete for the purposes of fire spread modelling (particularly those reliant on the curing formulas of dynamic fuel loading) until it is provided as a characteristic of local weather conditions.

The fuel models supported by GridFire include the standard 13 fuel models of Rothermel, Albini, and Anderson(Anderson, 1982) and the additional 40 fuel models defined by Scott and Burgan(Scott and Burgan, 2005). These are all concisely encoded in an internal data structure, which may be updated to include additional custom fuel models desired by the user.

```
(ns gridfire.fuel-models)

(def fuel-models
  "Lookup table including one entry for each of the Anderson 13 and
   Scott & Burgan 40 fuel models. The fields have the following
   meanings:
   {number
    [name delta M_x-dead h
     [w_o-dead-1hr w_o-dead-10hr w_o-dead-100hr w_o-live-herbaceous w_o-live-woody]
     [sigma-dead-1hr sigma-dead-10hr sigma-dead-100hr sigma-live-herbaceous sigma-live-woody]]
   }"
  {
   ;; Grass and Grass-dominated (short-grass,timber-grass-and-understory,tall-grass)
   1   [:R01 1.0 12 8 [0.0340 0.0000 0.0000 0.0000 0.0000] [3500.0    0.0  0.0    0.0    0.0]]
   2   [:R02 1.0 15 8 [0.0920 0.0460 0.0230 0.0230 0.0000] [3000.0 109.0 30.0 1500.0    0.0]]
   3   [:R03 2.5 25 8 [0.1380 0.0000 0.0000 0.0000 0.0000] [1500.0    0.0  0.0    0.0    0.0]]
   ;; Chaparral and Shrubfields (chaparral,brush,dormant-brush-hardwood-slash,southern-rough)
   4   [:R04 6.0 20 8 [0.2300 0.1840 0.0920 0.2300 0.0000] [2000.0 109.0 30.0 1500.0    0.0]]
```

```
5   [:R05 2.0 20 8 [0.0460 0.0230 0.0000 0.0920 0.0000] [2000.0 109.0  0.0 1500.0    0.0]]
6   [:R06 2.5 25 8 [0.0690 0.1150 0.0920 0.0000 0.0000] [1750.0 109.0 30.0    0.0    0.0]]
7   [:R07 2.5 40 8 [0.0520 0.0860 0.0690 0.0170 0.0000] [1750.0 109.0 30.0 1550.0    0.0]]
;; Timber Litter (closed-timber-litter,hardwood-litter,timber-litter-and-understory)
8   [:R08 0.2 30 8 [0.0690 0.0460 0.1150 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
9   [:R09 0.2 25 8 [0.1340 0.0190 0.0070 0.0000 0.0000] [2500.0 109.0 30.0    0.0    0.0]]
10  [:R10 1.0 25 8 [0.1380 0.0920 0.2300 0.0920 0.0000] [2000.0 109.0 30.0 1500.0    0.0]]
;; Logging Slash (light-logging-slash,medium-logging-slash,heavy-logging-slash)
11  [:R11 1.0 15 8 [0.0690 0.2070 0.2530 0.0000 0.0000] [1500.0 109.0 30.0    0.0    0.0]]
12  [:R12 2.3 20 8 [0.1840 0.6440 0.7590 0.0000 0.0000] [1500.0 109.0 30.0    0.0    0.0]]
13  [:R13 3.0 25 8 [0.3220 1.0580 1.2880 0.0000 0.0000] [1500.0 109.0 30.0    0.0    0.0]]
;; Nonburnable (NB)
91  [:NB1 0.0  0 0 [0.0000 0.0000 0.0000 0.0000 0.0000] [   0.0   0.0  0.0    0.0    0.0]]
92  [:NB2 0.0  0 0 [0.0000 0.0000 0.0000 0.0000 0.0000] [   0.0   0.0  0.0    0.0    0.0]]
93  [:NB3 0.0  0 0 [0.0000 0.0000 0.0000 0.0000 0.0000] [   0.0   0.0  0.0    0.0    0.0]]
98  [:NB4 0.0  0 0 [0.0000 0.0000 0.0000 0.0000 0.0000] [   0.0   0.0  0.0    0.0    0.0]]
99  [:NB5 0.0  0 0 [0.0000 0.0000 0.0000 0.0000 0.0000] [   0.0   0.0  0.0    0.0    0.0]]
;; Grass (GR)
101 [:GR1 0.4 15 8 [0.0046 0.0000 0.0000 0.0138 0.0000] [2200.0 109.0 30.0 2000.0    0.0]]
102 [:GR2 1.0 15 8 [0.0046 0.0000 0.0000 0.0459 0.0000] [2000.0 109.0 30.0 1800.0    0.0]]
103 [:GR3 2.0 30 8 [0.0046 0.0184 0.0000 0.0689 0.0000] [1500.0 109.0 30.0 1300.0    0.0]]
104 [:GR4 2.0 15 8 [0.0115 0.0000 0.0000 0.0872 0.0000] [2000.0 109.0 30.0 1800.0    0.0]]
105 [:GR5 1.5 40 8 [0.0184 0.0000 0.0000 0.1148 0.0000] [1800.0 109.0 30.0 1600.0    0.0]]
106 [:GR6 1.5 40 9 [0.0046 0.0000 0.0000 0.1561 0.0000] [2200.0 109.0 30.0 2000.0    0.0]]
107 [:GR7 3.0 15 8 [0.0459 0.0000 0.0000 0.2479 0.0000] [2000.0 109.0 30.0 1800.0    0.0]]
108 [:GR8 4.0 30 8 [0.0230 0.0459 0.0000 0.3352 0.0000] [1500.0 109.0 30.0 1300.0    0.0]]
109 [:GR9 5.0 40 8 [0.0459 0.0459 0.0000 0.4132 0.0000] [1800.0 109.0 30.0 1600.0    0.0]]
;; Grass-Shrub (GS)
121 [:GS1 0.9 15 8 [0.0092 0.0000 0.0000 0.0230 0.0298] [2000.0 109.0 30.0 1800.0 1800.0]]
122 [:GS2 1.5 15 8 [0.0230 0.0230 0.0000 0.0275 0.0459] [2000.0 109.0 30.0 1800.0 1800.0]]
123 [:GS3 1.8 40 8 [0.0138 0.0115 0.0000 0.0666 0.0574] [1800.0 109.0 30.0 1600.0 1600.0]]
124 [:GS4 2.1 40 8 [0.0872 0.0138 0.0046 0.1561 0.3260] [1800.0 109.0 30.0 1600.0 1600.0]]
;; Shrub (SH)
141 [:SH1 1.0 15 8 [0.0115 0.0115 0.0000 0.0069 0.0597] [2000.0 109.0 30.0 1800.0 1600.0]]
142 [:SH2 1.0 15 8 [0.0620 0.1102 0.0344 0.0000 0.1768] [2000.0 109.0 30.0    0.0 1600.0]]
143 [:SH3 2.4 40 8 [0.0207 0.1377 0.0000 0.0000 0.2847] [1600.0 109.0 30.0    0.0 1400.0]]
144 [:SH4 3.0 30 8 [0.0390 0.0528 0.0092 0.0000 0.1171] [2000.0 109.0 30.0 1800.0 1600.0]]
145 [:SH5 6.0 15 8 [0.1653 0.0964 0.0000 0.0000 0.1331] [ 750.0 109.0 30.0    0.0 1600.0]]
146 [:SH6 2.0 30 8 [0.1331 0.0666 0.0000 0.0000 0.0643] [ 750.0 109.0 30.0    0.0 1600.0]]
147 [:SH7 6.0 15 8 [0.1607 0.2433 0.1010 0.0000 0.1561] [ 750.0 109.0 30.0    0.0 1600.0]]
148 [:SH8 3.0 40 8 [0.0941 0.1561 0.0390 0.0000 0.1997] [ 750.0 109.0 30.0    0.0 1600.0]]
149 [:SH9 4.4 40 8 [0.2066 0.1125 0.0000 0.0712 0.3214] [ 750.0 109.0 30.0 1800.0 1500.0]]
;; Timber-Understory (TU)
161 [:TU1 0.6 20 8 [0.0092 0.0413 0.0689 0.0092 0.0413] [2000.0 109.0 30.0 1800.0 1600.0]]
162 [:TU2 1.0 30 8 [0.0436 0.0826 0.0574 0.0000 0.0092] [2000.0 109.0 30.0    0.0 1600.0]]
163 [:TU3 1.3 30 8 [0.0505 0.0069 0.0115 0.0298 0.0505] [1800.0 109.0 30.0 1600.0 1400.0]]
164 [:TU4 0.5 12 8 [0.2066 0.0000 0.0000 0.0000 0.0918] [2300.0 109.0 30.0    0.0 2000.0]]
165 [:TU5 1.0 25 8 [0.1837 0.1837 0.1377 0.0000 0.1377] [1500.0 109.0 30.0    0.0  750.0]]
;; Timber Litter (TL)
181 [:TL1 0.2 30 8 [0.0459 0.1010 0.1653 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
```

```
182 [:TL2 0.2 25 8 [0.0643 0.1056 0.1010 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
183 [:TL3 0.3 20 8 [0.0230 0.1010 0.1286 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
184 [:TL4 0.4 25 8 [0.0230 0.0689 0.1928 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
185 [:TL5 0.6 25 8 [0.0528 0.1148 0.2020 0.0000 0.0000] [2000.0 109.0 30.0    0.0 1600.0]]
186 [:TL6 0.3 25 8 [0.1102 0.0551 0.0551 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
187 [:TL7 0.4 25 8 [0.0138 0.0643 0.3719 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
188 [:TL8 0.3 35 8 [0.2663 0.0643 0.0505 0.0000 0.0000] [1800.0 109.0 30.0    0.0    0.0]]
189 [:TL9 0.6 35 8 [0.3053 0.1515 0.1905 0.0000 0.0000] [1800.0 109.0 30.0    0.0 1600.0]]
;; Slash-Blowdown (SB)
201 [:SB1 1.0 25 8 [0.0689 0.1377 0.5051 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
202 [:SB2 1.0 25 8 [0.2066 0.1951 0.1837 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
203 [:SB3 1.2 25 8 [0.2525 0.1263 0.1377 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
204 [:SB4 2.7 25 8 [0.2410 0.1607 0.2410 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
})
```

Once fuel moisture is added to the base fuel model definitions, they will each contain values for the following six fuel size classes:

1. Dead 1 hour (< 1/4" diameter)
2. Dead 10 hour (1/4"–1" diameter)
3. Dead 100 hour (1"–3" diameter)
4. Dead herbaceous (dynamic fuel models only)
5. Live herbaceous
6. Live woody

In order to more easily encode mathematical operations over these size classes, we define a collection of utility functions that will later be used in both the fuel moisture and fire spread algorithms.

```
(defn map-category [f]
  {:dead (f :dead) :live (f :live)})

(defn map-size-class [f]
  {:dead {:1hr        (f :dead :1hr)
          :10hr       (f :dead :10hr)
          :100hr      (f :dead :100hr)
          :herbaceous (f :dead :herbaceous)}
   :live {:herbaceous (f :live :herbaceous)
          :woody      (f :live :woody)}})

(defn category-sum [f]
  (+ (f :dead) (f :live)))

(defn size-class-sum [f]
  {:dead (+ (f :dead :1hr) (f :dead :10hr) (f :dead :100hr) (f :dead :herbaceous))
   :live (+ (f :live :herbaceous) (f :live :woody))})
```

Using these new size class processing functions, we can translate the encoded fuel model definitions into human-readable representations of the fuel model properties.

```
(defn build-fuel-model
  [fuel-model-number]
  (let [[name delta M_x-dead h
         [w_o-dead-1hr w_o-dead-10hr w_o-dead-100hr
          w_o-live-herbaceous w_o-live-woody]
         [sigma-dead-1hr sigma-dead-10hr sigma-dead-100hr
          sigma-live-herbaceous sigma-live-woody]]
        (fuel-models fuel-model-number)
        M_x-dead (* M_x-dead 0.01)
        h        (* h 1000.0)]
    {:name   name
     :number fuel-model-number
     :delta  delta
     :M_x    {:dead {:1hr        M_x-dead
                     :10hr       M_x-dead
                     :100hr      M_x-dead
                     :herbaceous 0.0}
              :live {:herbaceous 0.0
                     :woody      0.0}}
     :w_o    {:dead {:1hr        w_o-dead-1hr
                     :10hr       w_o-dead-10hr
                     :100hr      w_o-dead-100hr
                     :herbaceous 0.0}
              :live {:herbaceous w_o-live-herbaceous
                     :woody      w_o-live-woody}}
     :sigma  {:dead {:1hr        sigma-dead-1hr
                     :10hr       sigma-dead-10hr
                     :100hr      sigma-dead-100hr
                     :herbaceous 0.0}
              :live {:herbaceous sigma-live-herbaceous
                     :woody      sigma-live-woody}}
     :h      {:dead {:1hr        h
                     :10hr       h
                     :100hr      h
                     :herbaceous h}
              :live {:herbaceous h
                     :woody      h}}
     :rho_p  {:dead {:1hr        32.0
                     :10hr       32.0
                     :100hr      32.0
                     :herbaceous 32.0}
              :live {:herbaceous 32.0
                     :woody      32.0}}
     :S_T    {:dead {:1hr        0.0555
                     :10hr       0.0555
                     :100hr      0.0555
                     :herbaceous 0.0555}
              :live {:herbaceous 0.0555
                     :woody      0.0555}}
```

```
    :S_e    {:dead {:1hr        0.01
                   :10hr       0.01
                   :100hr      0.01
                   :herbaceous 0.01}
             :live {:herbaceous 0.01
                   :woody      0.01}}}))
```

Although most fuel model properties are static with respect to environmental conditions, the fuel moisture content can have two significant impacts on a fuel model's burning potential:

1. Dynamic fuel loading
2. Live moisture of extinction

These two topics are discussed in the remainder of this section.

5.1.1 Dynamic Fuel Loading

All of the Scott & Burgan 40 fuel models with a live herbaceous component are considered dynamic. In these models, a fraction of the live herbaceous load is transferred to a new dead herbaceous category as a function of live herbaceous moisture content (see equation below).(Burgan, 1979) The dead herbaceous category uses the dead 1 hour moisture content, dead moisture of extinction, and live herbaceous surface-area-to-volume-ratio. In the following formula, M_f^{lh} is the live herbaceous moisture content.

$$\text{FractionGreen} = \begin{cases} 0 & M_f^{lh} \leq 0.3 \\ 1 & M_f^{lh} \geq 1.2 \\ \frac{M_f^{lh}}{0.9} - \frac{1}{3} & \text{else} \end{cases}$$
$$\text{FractionCured} = 1 - \text{FractionGreen}$$

```
(defn add-dynamic-fuel-loading
  [{:keys [number M_x M_f w_o sigma] :as fuel-model}]
  (let [live-herbaceous-load (-> w_o :live :herbaceous)]
    (if (and (> number 100) (pos? live-herbaceous-load))
      ;; dynamic fuel model
      (let [fraction-green (max 0.0 (min 1.0 (- (/ (-> M_f :live :herbaceous) 0.9) 1/3)))
            fraction-cured (- 1.0 fraction-green)]
        (-> fuel-model
            (assoc-in [:M_f   :dead :herbaceous] (-> M_f :dead :1hr))
            (assoc-in [:M_x   :dead :herbaceous] (-> M_x :dead :1hr))
            (assoc-in [:w_o   :dead :herbaceous] (* live-herbaceous-load fraction-cured))
            (assoc-in [:w_o   :live :herbaceous] (* live-herbaceous-load fraction-green))
            (assoc-in [:sigma :dead :herbaceous] (-> sigma :live :herbaceous))))
      ;; static fuel model
      fuel-model)))
```

Once the dynamic fuel loading is applied, we can compute the size class weighting factors expressed in equations 53-57 in Rothermel 1972(Rothermel, 1972). For brevity, these formulas are elided from this text.

```
(defn add-weighting-factors
  [{:keys [w_o sigma rho_p] :as fuel-model}]
  (let [A_ij (map-size-class (fn [i j] (/ (* (-> sigma i j) (-> w_o i j))
                                          (-> rho_p i j))))

        A_i  (size-class-sum (fn [i j] (-> A_ij i j)))

        A_T  (category-sum (fn [i] (-> A_i i)))

        f_ij (map-size-class (fn [i j] (if (pos? (-> A_i i))
                                         (/ (-> A_ij i j)
                                            (-> A_i i))
                                         0.0)))

        f_i  (map-category (fn [i] (if (pos? A_T)
                                     (/ (-> A_i i) A_T)
                                     0.0)))

        firemod-size-classes (map-size-class
                              (fn [i j] (condp <= (-> sigma i j)
                                          1200 1
                                          192  2
                                          96   3
                                          48   4
                                          16   5
                                          0    6)))

        firemod-weights (into {}
                              (for [[category size-classes] firemod-size-classes]
                                [category
                                 (apply merge-with +
                                        (for [[size-class firemod-size-class] size-classes]
                                          {firemod-size-class (get-in f_ij [category size-class])}))]))

        g_ij (map-size-class (fn [i j]
                               (let [firemod-size-class (-> firemod-size-classes i j)]
                                 (get-in firemod-weights [i firemod-size-class]))))]
    (-> fuel-model
        (assoc :f_ij f_ij)
        (assoc :f_i  f_i)
        (assoc :g_ij g_ij))))
```

5.1.2 Live Moisture of Extinction

The live moisture of extinction for each fuel model is determined from the dead fuel moisture content, the dead moisture of extinction, and the ratio of dead fuel loading to live fuel loading using Equation 88 from Rothermel 1972, adjusted according to Albini 1976 Appendix III to match the behavior of Albini's original FIREMODS library.(Rothermel, 1972, Albini, 1976) Whenever the fuel moisture content becomes greater than or equal to the moisture of extinction, a fire will no longer spread through that fuel. Here are the formulas referenced above:

$$M_x^l = \max(M_x^d, 2.9\,W'\,(1 - \frac{M_f^d}{M_x^d}) - 0.226)$$

$$W' = \frac{\sum_{c \in D} w_o^c\, e^{-138/\sigma^c}}{\sum_{c \in L} w_o^c\, e^{-500/\sigma^c}}$$

$$M_f^d = \frac{\sum_{c \in D} w_o^c\, M_f^c\, e^{-138/\sigma^c}}{\sum_{c \in D} w_o^c\, e^{-138/\sigma^c}}$$

where M_x^l is the live moisture of extinction, M_x^d is the dead moisture of extinction, D is the set of dead fuel size classes (1hr, 10hr, 100hr, herbaceous), L is the set of live fuel size classes (herbaceous, woody), w_o^c is the dry weight loading of size class c, σ^c is the surface area to volume ratio of size class c, and M_f^c is the moisture content of size class c.

```
(defn add-live-moisture-of-extinction
  "Equation 88 from Rothermel 1972 adjusted by Albini 1976 Appendix III."
  [{:keys [w_o sigma M_f M_x] :as fuel-model}]
  (let [dead-loading-factor  (:dead (size-class-sum
                                     (fn [i j] (if (pos? (-> sigma i j))
                                                 (* (-> w_o i j)
                                                    (Math/exp (/ -138.0 (-> sigma i j))))
                                                 0.0))))
        live-loading-factor  (:live (size-class-sum
                                     (fn [i j] (if (pos? (-> sigma i j))
                                                 (* (-> w_o i j)
                                                    (Math/exp (/ -500.0 (-> sigma i j))))
                                                 0.0))))
        dead-moisture-factor (:dead (size-class-sum
                                     (fn [i j] (if (pos? (-> sigma i j))
                                                 (* (-> w_o i j)
                                                    (Math/exp (/ -138.0 (-> sigma i j)))
                                                    (-> M_f i j))
                                                 0.0))))
        dead-to-live-ratio   (if (pos? live-loading-factor)
                               (/ dead-loading-factor live-loading-factor))
        dead-fuel-moisture   (if (pos? dead-loading-factor)
                               (/ dead-moisture-factor dead-loading-factor)
```

```
                              0.0)
        M_x-dead            (-> M_x :dead :1hr)
        M_x-live            (if (pos? live-loading-factor)
                              (max M_x-dead
                                   (- (* 2.9
                                         dead-to-live-ratio
                                         (- 1.0 (/ dead-fuel-moisture M_x-dead)))
                                      0.226))
                              M_x-dead)]
    (-> fuel-model
        (assoc-in [:M_x :live :herbaceous] M_x-live)
        (assoc-in [:M_x :live :woody]      M_x-live))))

(defn moisturize
  [fuel-model fuel-moisture]
  (-> fuel-model
      (assoc :M_f fuel-moisture)
      (assoc-in [:M_f :dead :herbaceous] 0.0)
      (add-dynamic-fuel-loading)
      (add-weighting-factors)
      (add-live-moisture-of-extinction)))
```

This concludes our coverage of fuel models and and fuel moisture.

5.2 Surface Fire Formulas

To simulate fire behavior in as similar a way as possible to the US government-sponsored fire models (e.g., FARSITE, FlamMap, FPA, BehavePlus), we adopt the surface fire spread and reaction intensity formulas from Rothermel's 1972 publication "A Mathematical Model for Predicting Fire Spread in Wildland Fuels".(Rothermel, 1972)

Very briefly, the surface rate of spread of a fire's leading edge R is described by the following formula:

$$R = \frac{I_R \, \xi \, (1 + \phi_W + \phi_S)}{\rho_b \, \epsilon \, Q_{ig}}$$

where these terms have the meanings shown in Table 3.

For a full description of each of the subcomponents of Rothermel's surface fire spread rate equation, see the Rothermel 1972 reference above. In addition to applying the base Rothermel equations, GridFire reduces the spread rates for all of the Scott & Burgan 40 fuel models of the grass subgroup (101-109) by 50%. This addition was originally suggested by Chris Lautenberger of REAX Engineering.

For efficiency, the surface fire spread equation given above is computed first without introducing the effects of wind and slope ($\phi_W = \phi_S = 0$).

Table 3: Inputs to Rothermel's surface fire rate of spread equation

Term	Meaning
R	surface fire spread rate
I_R	reaction intensity
ξ	propagating flux ratio
ϕ_W	wind coefficient
ϕ_S	slope factor
ρ_b	oven-dry fuel bed bulk density
ϵ	effective heating number
Q_{ig}	heat of preignition

```
(ns gridfire.surface-fire
  (:require [gridfire.fuel-models :refer [map-category map-size-class
                                          category-sum size-class-sum]]))

(def grass-fuel-model? #(and (> % 100) (< % 110)))

(defn rothermel-surface-fire-spread-no-wind-no-slope
  "Returns the rate of surface fire spread in ft/min and the reaction
   intensity (i.e., amount of heat output) of a fire in Btu/ft^2*min
   given a map containing these keys:
   - number [fuel model number]
   - delta [fuel depth (ft)]
   - w_o [ovendry fuel loading (lb/ft^2)]
   - sigma [fuel particle surface-area-to-volume ratio (ft^2/ft^3)]
   - h [fuel particle low heat content (Btu/lb)]
   - rho_p [ovendry particle density (lb/ft^3)]
   - S_T [fuel particle total mineral content (lb minerals/lb ovendry wood)]
   - S_e [fuel particle effective mineral content (lb silica-free minerals/lb ovendry wood)]
   - M_x [moisture content of extinction (lb moisture/lb ovendry wood)]
   - M_f [fuel particle moisture content (lb moisture/lb ovendry wood)]
   - f_ij [percent of load per size class (%)]
   - f_i [percent of load per category (%)]
   - g_ij [percent of load per size class from Albini_1976_FIREMOD, page 20]"
  [{:keys [number delta w_o sigma h rho_p S_T S_e M_x M_f f_ij f_i g_ij] :as fuel-model}]
  (let [S_e_i      (size-class-sum (fn [i j] (* (-> f_ij i j) (-> S_e i j))))

        ;; Mineral damping coefficient
        eta_S_i    (map-category (fn [i] (let [S_e_i (-> S_e_i i)]
                                           (if (pos? S_e_i)
                                             (/ 0.174 (Math/pow S_e_i 0.19))
                                             1.0))))

        M_f_i      (size-class-sum (fn [i j] (* (-> f_ij i j) (-> M_f i j))))

        M_x_i      (size-class-sum (fn [i j] (* (-> f_ij i j) (-> M_x i j))))
```

```
r_M_i       (map-category (fn [i] (let [M_f (-> M_f_i i)
                                        M_x (-> M_x_i i)]
                                    (if (pos? M_x)
                                      (min 1.0 (/ M_f M_x))
                                      1.0))))

;; Moisture damping coefficient
eta_M_i     (map-category (fn [i] (+ 1.0
                                     (* -2.59 (-> r_M_i i))
                                     (* 5.11 (Math/pow (-> r_M_i i) 2))
                                     (* -3.52 (Math/pow (-> r_M_i i) 3)))))

h_i         (size-class-sum (fn [i j] (* (-> f_ij i j) (-> h i j))))

;; Net fuel loading (lb/ft^2)
W_n_i       (size-class-sum (fn [i j] (* (-> g_ij i j)
                                         (-> w_o i j)
                                         (- 1.0 (-> S_T i j)))))

beta_i      (size-class-sum (fn [i j] (/ (-> w_o i j) (-> rho_p i j))))

;; Packing ratio
beta        (if (pos? delta)
              (/ (category-sum (fn [i] (-> beta_i i))) delta)
              0.0)

sigma'_i    (size-class-sum (fn [i j] (* (-> f_ij i j) (-> sigma i j))))

sigma'      (category-sum (fn [i] (* (-> f_i i) (-> sigma'_i i))))

;; Optimum packing ratio
beta_op     (if (pos? sigma')
              (/ 3.348 (Math/pow sigma' 0.8189))
              1.0)

;; Albini 1976 replaces (/ 1 (- (* 4.774 (Math/pow sigma' 0.1)) 7.27))
A           (if (pos? sigma')
              (/ 133.0 (Math/pow sigma' 0.7913))
              0.0)

;; Maximum reaction velocity (1/min)
Gamma'_max (/ (Math/pow sigma' 1.5)
               (+ 495.0 (* 0.0594 (Math/pow sigma' 1.5))))

;; Optimum reaction velocity (1/min)
Gamma'      (* Gamma'_max
               (Math/pow (/ beta beta_op) A)
               (Math/exp (* A (- 1.0 (/ beta beta_op)))))
```

```
;; Reaction intensity (Btu/ft^2*min)
I_R         (* Gamma' (category-sum (fn [i] (* (W_n_i i) (h_i i)
                                               (eta_M_i i) (eta_S_i i)))))

;; Propagating flux ratio
xi          (/ (Math/exp (* (+ 0.792 (* 0.681 (Math/pow sigma' 0.5)))
                            (+ beta 0.1)))
               (+ 192.0 (* 0.2595 sigma')))

E           (* 0.715 (Math/exp (* -3.59 (/ sigma' 10000.0))))

B           (* 0.02526 (Math/pow sigma' 0.54))

C           (* 7.47 (Math/exp (* -0.133 (Math/pow sigma' 0.55))))

;; Derive wind factor
get-phi_W   (fn [midflame-wind-speed]
              (if (and (pos? beta) (pos? midflame-wind-speed))
                (-> midflame-wind-speed
                    (Math/pow B)
                    (* C)
                    (/ (Math/pow (/ beta beta_op) E)))
                0.0))

;; Derive wind speed from wind factor
get-wind-speed (fn [phi_W]
                 (-> phi_W
                     (* (Math/pow (/ beta beta_op) E))
                     (/ C)
                     (Math/pow (/ 1.0 B))))

;; Derive slope factor
get-phi_S   (fn [slope]
              (if (and (pos? beta) (pos? slope))
                (* 5.275 (Math/pow beta -0.3) (Math/pow slope 2.0))
                0.0))

;; Heat of preignition (Btu/lb)
Q_ig        (map-size-class (fn [i j] (+ 250.0 (* 1116.0 (-> M_f i j)))))

foo_i       (size-class-sum (fn [i j] (let [sigma_ij (-> sigma i j)
                                            Q_ig_ij  (-> Q_ig  i j)]
                                        (if (pos? sigma_ij)
                                          (* (-> f_ij i j)
                                             (Math/exp (/ -138 sigma_ij))
                                             Q_ig_ij)
                                          0.0))))

rho_b_i     (size-class-sum (fn [i j] (-> w_o i j)))
```

```
        ;; Ovendry bulk density (lb/ft^3)
        rho_b       (if (pos? delta)
                      (/ (category-sum (fn [i] (-> rho_b_i i))) delta)
                      0.0)

        rho_b-epsilon-Q_ig (* rho_b (category-sum (fn [i] (* (-> f_i i) (-> foo_i i)))))

        ;; Surface fire spread rate (ft/min)
        R           (if (pos? rho_b-epsilon-Q_ig)
                      (/ (* I_R xi) rho_b-epsilon-Q_ig)
                      0.0)

        ;; Addition proposed by Chris Lautenberger (REAX 2015)
        spread-rate-multiplier (if (grass-fuel-model? number) 0.5 1.0)]

    {:spread-rate        (* R spread-rate-multiplier)
     :reaction-intensity I_R
     :residence-time     (/ 384.0 sigma')
     :get-phi_W          get-phi_W
     :get-phi_S          get-phi_S
     :get-wind-speed     get-wind-speed}))
```

Later, this no-wind-no-slope value is used to compute the maximum spread rate and direction for the leading edge of the surface fire under analysis. Since Rothermel's original equations assume that the wind direction and slope are aligned, the effects of cross-slope winds must be taken into effect. Like Morais' HFire system, GridFire implements the vector addition procedure defined in Rothermel 1983 that combines the wind-only and slope-only spread rates independently to calculate the effective fire spread direction and magnitude.(Peterson et al., 2011, 2009, Morais, 2001, Rothermel, 1983)

A minor wrinkle is introduced when putting these calculations into practice because Rothermel's formulas all expect a measure of midflame wind speed. However, wind speed data is often collected at a height 20 feet above either unsheltered ground or a tree canopy layer if present. To convert this 20-ft wind speed to the required midflame wind speed value, GridFire uses the **wind adjustment factor** formula from Albini & Baughman 1979, parameterized as in BehavePlus, FARSITE, FlamMap, FSPro, and FPA according to Andrews 2012(Albini and Baughman, 1979, Andrews et al., 2012). This formula is shown below:

$$WAF = \begin{cases} \frac{1.83}{\ln(\frac{20.0+0.36FBD}{0.13FBD})} & CC = 0 \\ \frac{0.555}{\sqrt{(CH(CC/300.0))}\ln(\frac{20+0.36CH}{0.13CH})} & CC > 0 \end{cases}$$

where WAF is the unitless wind adjustment factor, FBD is the fuel bed depth in feet, CH is the canopy height in ft, and CC is the canopy cover percentage (0-100).

```
(defn wind-adjustment-factor
  "ft ft 0-100"
```

```
  [fuel-bed-depth canopy-height canopy-cover]
  (cond
    ;; sheltered: equation 2 based on CC and CH, CR=1 (Andrews 2012)
    (and (pos? canopy-cover)
         (pos? canopy-height))
    (/ 0.555 (* (Math/sqrt (* (/ canopy-cover 300.0) canopy-height))
                (Math/log (/ (+ 20.0 (* 0.36 canopy-height)) (* 0.13 canopy-height)))))

    ;; unsheltered: equation 6 H_F = H (Andrews 2012)
    (pos? fuel-bed-depth)
    (/ 1.83 (Math/log (/ (+ 20.0 (* 0.36 fuel-bed-depth)) (* 0.13 fuel-bed-depth))))

    ;; non-burnable fuel model
    :otherwise
    0.0))

(defn wind-adjustment-factor-elmfire
  "ft m 0-1"
  [fuel-bed-depth canopy-height canopy-cover]
  (cond
    ;; sheltered WAF
    (and (pos? canopy-cover)
         (pos? canopy-height))
    (* (/ 1.0 (Math/log (/ (+ 20.0 (* 0.36 (/ canopy-height 0.3048)))
                           (* 0.13 (/ canopy-height 0.3048)))))
       (/ 0.555 (Math/sqrt (* (/ canopy-cover 3.0) (/ canopy-height 0.3048)))))

    ;; unsheltered WAF
    (pos? fuel-bed-depth)
    (* (/ (+ 1.0 (/ 0.36 1.0))
          (Math/log (/ (+ 20.0 (* 0.36 fuel-bed-depth))
                       (* 0.13 fuel-bed-depth))))
       (- (Math/log (/ (+ 1.0 0.36) 0.13)) 1.0))

    ;; non-burnable fuel model
    :otherwise
    0.0))
```

The midflame wind speed that would be required to produce the combined spread rate in a no-slope scenario is termed the effective windspeed U_{eff}. Following the recommendations given in Appendix III of Albini 1976, these midflame wind speeds are all limited to $0.9I_R$.(Albini, 1976)

Next, the effective wind speed is used to compute the length to width ratio $\frac{L}{W}$ of an ellipse that approximates the fire front using equation 9 from Rothermel 1991.(Rothermel, 1991) This length to width ratio is then converted into an eccentricity measure of the ellipse using equation 8 from Albini and Chase 1980.(Albini and Chase, 1980) Finally, this eccentricity E is used to project the maximum spread rate to any point along the fire front. Here are the formulas used:

$$\frac{L}{W} = 1 + 0.002840909\, U_{\text{eff}}\, \text{EAF}$$
$$E = \frac{\sqrt{(\frac{L}{W})^2 - 1}}{\frac{L}{W}}$$
$$R_\theta = R_{\max} \left(\frac{1 - E}{1 - E\cos\theta} \right)$$

where θ is the angular offset from the direction of maximum fire spread, $\mathrm{R}_{\max}$ is the maximum spread rate, R_θ is the spread rate in direction θ, and EAF is the ellipse adjustment factor, a term introduced by Marco Morais and Seth Peterson in their HFire work that can be increased or decreased to make the fire shape more elliptical or circular respectively.(Peterson et al., 2009)

Note: The coefficient 0.002840909 in the $\frac{L}{W}$ formula is in units of min/ft. The original equation from Rothermel 1991 used 0.25 in units of hr/mi, so this was converted to match GridFire's use of ft/min for U_{eff}.

```
(defn almost-zero? [^double x]
  (< (Math/abs x) 0.000001))

(defn degrees-to-radians
  [degrees]
  (/ (* degrees Math/PI) 180.0))

(defn radians-to-degrees
  [radians]
  (/ (* radians 180.0) Math/PI))

(defn scale-spread-to-max-wind-speed
  [{:keys [effective-wind-speed max-spread-direction] :as spread-properties}
   spread-rate max-wind-speed phi-max]
  (if (> effective-wind-speed max-wind-speed)
    {:max-spread-rate      (* spread-rate (+ 1.0 phi-max))
     :max-spread-direction max-spread-direction
     :effective-wind-speed max-wind-speed}
    spread-properties))

(defn add-eccentricity
  [{:keys [effective-wind-speed] :as spread-properties} ellipse-adjustment-factor]
  (let [length-width-ratio (+ 1.0 (* 0.002840909
                                     effective-wind-speed
                                     ellipse-adjustment-factor))
        eccentricity       (/ (Math/sqrt (- (Math/pow length-width-ratio 2.0) 1.0))
                              length-width-ratio)]
    (assoc spread-properties :eccentricity eccentricity)))
```

```
(defn smallest-angle-between [theta1 theta2]
  (let [angle (Math/abs ^double (- theta1 theta2))]
    (if (> angle 180.0)
      (- 360.0 angle)
      angle)))

(defn rothermel-surface-fire-spread-max
  "Note: fire ellipse adjustment factor, < 1.0 = more circular, > 1.0 = more elliptical"
  [{:keys [spread-rate reaction-intensity get-phi_W get-phi_S get-wind-speed]}
   midflame-wind-speed wind-from-direction slope aspect ellipse-adjustment-factor]
  (let [phi_W             (get-phi_W midflame-wind-speed)
        phi_S             (get-phi_S slope)
        slope-direction   (mod (+ aspect 180.0) 360.0)
        wind-to-direction (mod (+ wind-from-direction 180.0) 360.0)
        max-wind-speed    (* 0.9 reaction-intensity)
        phi-max           (get-phi_W max-wind-speed)]
    (->
     (cond (and (almost-zero? midflame-wind-speed) (almost-zero? slope))
           ;; no wind, no slope
           {:max-spread-rate      spread-rate
            :max-spread-direction 0.0
            :effective-wind-speed 0.0}

           (almost-zero? slope)
           ;; wind only
           {:max-spread-rate      (* spread-rate (+ 1.0 phi_W))
            :max-spread-direction wind-to-direction
            :effective-wind-speed midflame-wind-speed}

           (almost-zero? midflame-wind-speed)
           ;; slope only
           {:max-spread-rate      (* spread-rate (+ 1.0 phi_S))
            :max-spread-direction slope-direction
            :effective-wind-speed (get-wind-speed phi_S)}

           (< (smallest-angle-between wind-to-direction slope-direction) 15.0)
           ;; wind blows (within 15 degrees of) upslope
           {:max-spread-rate      (* spread-rate (+ 1.0 phi_W phi_S))
            :max-spread-direction slope-direction
            :effective-wind-speed (get-wind-speed (+ phi_W phi_S))}

           :else
           ;; wind blows across slope
           (let [slope-magnitude    (* spread-rate phi_S)
                 wind-magnitude     (* spread-rate phi_W)
                 difference-angle   (degrees-to-radians
                                     (mod (- wind-to-direction slope-direction) 360.0))
                 x                  (+ slope-magnitude
                                       (* wind-magnitude (Math/cos difference-angle)))
```

```
                  y                    (* wind-magnitude (Math/sin difference-angle))
                  combined-magnitude (Math/sqrt (+ (* x x) (* y y)))]
              (if (almost-zero? combined-magnitude)
                {:max-spread-rate      spread-rate
                 :max-spread-direction 0.0
                 :effective-wind-speed 0.0}
                (let [max-spread-rate      (+ spread-rate combined-magnitude)
                      phi-combined         (- (/ max-spread-rate spread-rate) 1.0)
                      offset               (radians-to-degrees
                                            (Math/asin (/ (Math/abs y) combined-magnitude)))
                      offset'              (if (>= x 0.0)
                                             (if (>= y 0.0)
                                               offset
                                               (- 360.0 offset))
                                             (if (>= y 0.0)
                                               (- 180.0 offset)
                                               (+ 180.0 offset)))
                      max-spread-direction (mod (+ slope-direction offset') 360.0)
                      effective-wind-speed (get-wind-speed phi-combined)]
                  {:max-spread-rate      max-spread-rate
                   :max-spread-direction max-spread-direction
                   :effective-wind-speed effective-wind-speed}))))
      (scale-spread-to-max-wind-speed spread-rate max-wind-speed phi-max)
      (add-eccentricity ellipse-adjustment-factor))))

(defn rothermel-surface-fire-spread-any
  [{:keys [max-spread-rate max-spread-direction eccentricity]} spread-direction]
  (let [theta (smallest-angle-between max-spread-direction spread-direction)]
    (if (or (almost-zero? eccentricity) (almost-zero? theta))
      max-spread-rate
      (* max-spread-rate (/ (- 1.0 eccentricity)
                            (- 1.0 (* eccentricity
                                      (Math/cos (degrees-to-radians theta))))))))))
```

Using these surface fire spread rate and reaction intensity values, we next calculate fire intensity values by applying Anderson's flame depth formula and Byram's fire line intensity and flame length equations as described below.(Anderson, 1969, Byram, 1959)

$$t = \frac{384}{\sigma}$$
$$D = Rt$$
$$I = \frac{I_R D}{60}$$
$$L = 0.45(I)^{0.46}$$

where σ is the weighted sum by size class of the fuel model's surface area to volume ratio in ft^2/ft^3, t is the residence time in minutes, R is the surface fire spread rate in ft/min, D is the flame depth in ft, I_R is the reaction intensity in Btu/ft^2/min, I is the fire line intensity in Btu/ft/s, and L is the flame length in ft.

```
(defn anderson-flame-depth
  "Returns the depth, or front-to-back distance, of the actively flaming zone
   of a free-spreading fire in ft given:
   - spread-rate (ft/min)
   - residence-time (min)"
  [spread-rate residence-time]
  (* spread-rate residence-time))

(defn byram-fire-line-intensity
  "Returns the rate of heat release per unit of fire edge in Btu/ft*s given:
   - reaction-intensity (Btu/ft^2*min)
   - flame-depth (ft)"
  [reaction-intensity flame-depth]
  (/ (* reaction-intensity flame-depth) 60.0))

(defn byram-flame-length
  "Returns the average flame length in ft given:
   - fire-line-intensity (Btu/ft*s)"
  [fire-line-intensity]
  (* 0.45 (Math/pow fire-line-intensity 0.46)))
```

This concludes our coverage of the surface fire behavior equations implemented in GridFire. In Section 5.4, these formulas will be translated from one-dimension to two-dimensional spread on a raster grid. Before we move on to that, however, the following section explains how crown fire behavior metrics are incorporated into our model.

5.3 Crown Fire Formulas

In order to incorporate the effects of crown fire behavior, GridFire includes the crown fire initiation routine from Van Wagner 1977.(Wagner, 1977) According to this approach, there are two threshold values (*critical intensity* and *critical spread rate*) that must be calculated in order to determine whether a fire will become an active or passive crown fire or simply remain a surface fire. The formulas for these thresholds are as follows:

$$
\begin{aligned}
H &= 460 + 2600 M^f \\
I^* &= (0.01\, Z_b\, H)^{1.5} \\
R^* &= \frac{3.0}{B_m}
\end{aligned}
$$

where H is the heat of ignition for the herbaceous material in the canopy in kJ/kg, M^f is the foliar moisture content in lb moisture/lb ovendry weight, Z_b is the canopy base height in meters, I^* is the critical intensity in kW/m, B_m is the crown bulk density in kg/m^3, and R^* is the critical spread rate in m/min.

If the canopy cover is greater than 40% and the surface fire line intensity is greater than the critical intensity ($I > I^*$), then crown fire initiation occurs.

```
(ns gridfire.crown-fire)

(defn ft->m [ft] (* 0.3048 ft))

(defn kW-m->Btu-ft-s [kW-m] (* 0.288894658272 kW-m))

(defn van-wagner-crown-fire-initiation?
  "- canopy-cover (0-100 %)
   - canopy-base-height (ft)
   - foliar-moisture (lb moisture/lb ovendry weight)
   - fire-line-intensity (Btu/ft*s)"
  [canopy-cover canopy-base-height foliar-moisture fire-line-intensity]
  (and (> canopy-cover 40.0)
       (-> (+ 460.0 (* 2600.0 foliar-moisture)) ;; heat-of-ignition = kJ/kg
           (* 0.01 (ft->m canopy-base-height))
           (Math/pow 1.5) ;; critical-intensity = kW/m
           (kW-m->Btu-ft-s)
           (< fire-line-intensity))))
```

If crowning occurs, then the active and passive crown fire spread rates are calculated from the formulas given in Cruz 2005.(Cruz et al., 2005)

$$\mathrm{CROS}_A = 11.02\, U_{10m}^{0.90}\, B_m^{0.19}\, e^{-0.17\,\mathrm{EFFM}}$$
$$\mathrm{CROS}_P = \mathrm{CROS}_A\, e^{\frac{-\mathrm{CROS}_A}{R^*}}$$

where CROS_A is the active crown fire spread rate in m/min, U_{10m} is the 10 meter windspeed in km/hr, B_m is the crown bulk density in kg/m^3, EFFM is the estimated fine fuel moisture as a percent (0-100), and CROS_P is the passive crown fire spread rate in m/min.

If the active crown fire spread rate is greater than the critical spread rate ($\mathrm{CROS}_A > R^*$), then the crown fire will be active, otherwise passive.

```
(defn mph->km-hr [mph] (* 1.609344 mph))

(defn lb-ft3->kg-m3 [lb-ft3] (* 16.01846 lb-ft3))

(defn m->ft [m] (* 3.281 m))

(defn cruz-crown-fire-spread
  "Returns spread-rate in ft/min given:
   - wind-speed-20ft (mph)
```

```
  - crown-bulk-density (lb/ft^3)
  - estimated-fine-fuel-moisture (-> M_f :dead :1hr) (0-1)"
 [wind-speed-20ft crown-bulk-density estimated-fine-fuel-moisture]
 (let [wind-speed-10m               (/ (mph->km-hr wind-speed-20ft) 0.87) ;; km/hr
       crown-bulk-density           (lb-ft3->kg-m3 crown-bulk-density) ;; kg/m^3
       estimated-fine-fuel-moisture (* 100.0 estimated-fine-fuel-moisture)
       active-spread-rate           (* 11.02
                                       (Math/pow wind-speed-10m 0.90)
                                       (Math/pow crown-bulk-density 0.19)
                                       (Math/exp (* -0.17 estimated-fine-fuel-moisture)))
                                       ;; m/min
       critical-spread-rate         (/ 3.0 crown-bulk-density) ;; m/min
       criteria-for-active-crowning (/ active-spread-rate critical-spread-rate)]
   (m->ft
    (if (> active-spread-rate critical-spread-rate)
      active-spread-rate
      (* active-spread-rate (Math/exp (- criteria-for-active-crowning)))))))
```

Once the crown fire spread rate is determined, the crown fire line intensity and flame lengths may be derived using the following formulas:

$$I_c = \frac{R_c B (Z - Z_b) h}{60}$$
$$L_c = 0.45 (I + I_c)^{0.46}$$

where I_c is the crown fire line intensity in Btu/ft/s, R_c is the crown fire spread rate (either CROS_A or CROS_P) in ft/min, B is the crown bulk density in lb/ft^3, Z is the canopy height in ft, Z_b is the canopy base height in ft, h is the fuel model heat of combustion (generally 8000 Btu/lb), L_c is the crown flame length in ft, and I is the surface fire line intensity in Btu/ft/s.

```
;; heat of combustion is h from the fuel models (generally 8000 Btu/lb)
(defn crown-fire-line-intensity
  "(ft/min * lb/ft^3 * ft * Btu/lb)/60 = (Btu/ft*min)/60 = Btu/ft*s"
  [crown-spread-rate crown-bulk-density canopy-height canopy-base-height heat-of-combustion]
  (/ (* crown-spread-rate
        crown-bulk-density
        (- canopy-height canopy-base-height)
        heat-of-combustion)
     60.0))

(defn crown-fire-line-intensity-elmfire ;; kW/m
  [surface-fire-line-intensity crown-spread-rate crown-bulk-density
   canopy-height canopy-base-height]
  (let [heat-of-combustion 18000] ;; kJ/m^2
    (+ surface-fire-line-intensity ;; kW/m
       (/ (* 0.3048 ;; m/ft
             crown-spread-rate ;; ft/min
```

```
            crown-bulk-density ;; kg/m^3
            (- canopy-height canopy-base-height) ;; m
            heat-of-combustion) ;; kJ/kg
         60.0)))) ;; s/min
```

As with surface fire spread, the wind speed (this time the 20-ft wind speed in mph U_{20}) is used to compute the length to width ratio $\frac{L}{W}$ of an ellipse that approximates the crown fire front using equation 9 from Rothermel 1991.(Rothermel, 1991) This length to width ratio is then converted into an eccentricity measure of the ellipse using equation 8 from Albini and Chase 1980.(Albini and Chase, 1980) Finally, this eccentricity E is used to project the maximum spread rate to any point along the fire front. Here are the formulas used:

$$\frac{L}{W} = 1 + 0.125\, U_{20}\, \mathrm{EAF}$$

$$E = \frac{\sqrt{(\frac{L}{W})^2 - 1}}{\frac{L}{W}}$$

$$R_\theta = R_{\max} \left(\frac{1 - E}{1 - E \cos\theta} \right)$$

where θ is the angular offset from the direction of maximum fire spread, $\mathrm{R}_{\max}$ is the maximum spread rate, R_θ is the spread rate in direction θ, and EAF is the ellipse adjustment factor, a term introduced by Marco Morais and Seth Peterson in their HFire work that can be increased or decreased to make the fire shape more elliptical or circular respectively.(Peterson et al., 2009)

```
(defn crown-fire-eccentricity
  "mph"
  [wind-speed-20ft ellipse-adjustment-factor]
  (let [length-width-ratio (+ 1.0 (* 0.125
                                     wind-speed-20ft
                                     ellipse-adjustment-factor))]
    (/ (Math/sqrt (- (Math/pow length-width-ratio 2.0) 1.0))
       length-width-ratio)))

(defn elmfire-length-to-width-ratio
  "true/false mph int>0 ft/min
   Crown L/W = min(1.0 + 0.125*U20_mph, L/W_max)
   Surface L/W = 0.936*e^(0.2566*Ueff_mph) + 0.461*e^(-0.1548*Ueff_mph) - 0.397"
  [crown-fire? wind-speed-20ft max-length-to-width-ratio effective-wind-speed]
  (if crown-fire?
    (min (+ 1.0 (* 0.125 wind-speed-20ft)) max-length-to-width-ratio)
    (min (+ (* 0.936 (Math/exp (/ (* 0.2566 effective-wind-speed 60.0) 5280.0)))
            (* 0.461 (Math/exp (/ (* -0.1548 effective-wind-speed 60.0) 5280.0)))
```

```
        -0.397)
      8.0)))
```

This concludes our discussion of the crown fire behavior formulas used in GridFire.

5.4 Fire Spread on a Raster Grid

Although Rothermel's spread rate formula provides some useful insight into how quickly a fire's leading edge may travel, it offers no specific mechanism for simulating fire movement in two or more dimensions. Therefore, when attempting to use the Rothermel equations in any spatial analysis, one must begin by choosing a model of space and then decide how best to employ the spread rate equations along each possible burn trajectory.

In GridFire, SIG adopted a raster grid view of space so as to reduce the potentially exponential complexity of modeling a fractal shape (i.e., fire front) at high resolutions using vector approximation. This also provided the practical benefit of being able to work directly with widely used raster datasets, such as LANDFIRE, without a geometric lookup step or *a priori* translation to vector space.

In simulation tests versus FARSITE on several historical California fires, Marco Morais wrote that he saw similarly accurate results from both his HFire model and from FARSITE but experienced several orders of magnitude improvement in runtime efficiency.(Peterson et al., 2011, 2009, Morais, 2001) His explanation for this phenomenon was in the same vein as that described above, namely, that it was FARSITE's choice of vector space that slowed it down versus the faster raster-based HFire system.

Taking a cue from HFire's success in this regard, GridFire has adopted HFire's two-dimensional spread algorithm, called the *method of adaptive timesteps and fractional distances.* (Peterson et al., 2011, 2009, Morais, 2001) The following pseudo-code lays out the steps taken in this procedure:

1. Inputs

   (a) Read in the values shown in Table 4.

2. Initialization

   (a) Verify that **initial-ignition-site** and at least one of its neighboring cells has a burnable fuel model (not 91-99). Otherwise, terminate the simulation, indicating that no fire spread is possible.

   (b) Create three new matrices, called **fire-spread-matrix**, **flame-length-matrix**, and **fire-line-intensity-matrix**. All three are initialized to zero except for a value of 1 at the **initial-ignition-site**.

Table 4: Inputs to SIG's raster-based fire behavior model

Value	Units	Type
max-runtime	minutes	double
cell-size	feet	double
elevation-matrix	feet	core.matrix 2D double array
slope-matrix	vertical feet/horizontal feet	core.matrix 2D double array
aspect-matrix	degrees clockwise from north	core.matrix 2D double array
fuel-model-matrix	fuel model numbers 1-256	core.matrix 2D double array
canopy-height-matrix	feet	core.matrix 2D double array
canopy-base-height-matrix	feet	core.matrix 2D double array
crown-bulk-density-matrix	lb/ft^3	core.matrix 2D double array
canopy-cover-matrix	0-100	core.matrix 2D double array
wind-speed-20ft	miles/hour	double
wind-from-direction	degrees clockwise from North	double
fuel-moisture	%	map of doubles per fuel size class
foliar-moisture	%	double
ellipse-adjustment-factor	< 1.0 = circle, > 1.0 = ellipse	double
initial-ignition-site	point represented as [row col]	vector

(c) Set **global-clock** to 0. This will track the amount of time that has passed since the initial ignition in minutes.

(d) Create a new hash-map, called **ignited-cells**, which maps the **initial-ignition-site** to a set of trajectories into each of its burnable neighbors. See "Computing Burn Trajectories" below for the steps used in this procedure.

3. Computing Burn Trajectories

   (a) Look up the fuel model, slope, aspect, canopy height, canopy base height, crown bulk density, and canopy cover associated with the ignited cell in the input matrices.

   (b) Calculate the dead herbaceous size class parameters, live moisture of extinction, and size class weighting factors for this fuel model.

   (c) Use the Rothermel equations to calculate the minimum surface rate of spread (i.e., wind = slope = 0) leaving this cell.

   (d) Compute Albini and Baughman's wind adjustment factor for this cell using the fuel bed depth, canopy height, and canopy cover. Multiply this value by the 20-ft wind speed to derive the local midflame wind speed.

   (e) Calculate the maximum surface rate of spread (and bearing) originating from this cell using the Rothermel equations and taking into account the effects of downhill and cross-slope winds as described in Rothermel 1983.

   (f) Use the Cruz formulas to calculate the maximum crown fire spread rate from the 20-ft wind speed, crown bulk density, and dead 1-hr fuel moisture.

(g) Determine the surface and crown elliptical eccentricities by calculating their length-to-width ratios using the equations from Rothermel 1991.

(h) For each burnable neighboring cell:

i. Use the eccentricity values to determine the possible surface and crown rates of spread into it from the ignited cell.

ii. Compute Byram's surface fire line intensity and Rothermel's crown intensity from these spread rates.

iii. Apply Van Wagner's crown initiation model to determine if the fire will be a passive or active crown fire or remain a surface fire.

iv. In the surface fire case, the spread rate into this neighbor will simply be the surface spread rate calculated above. The fire line intensity is the surface fire line intensity, and the flame length is calculated from this intensity value using Byram's relation.

v. In the case of a crown fire, the spread rate into this neighbor will be the maximum of the surface and crown spread rates. The fire line intensity is the sum of the surface and crown intensities, and the flame length is once again computed from Byram's relation.

vi. Store this neighboring cell, the bearing to it from the ignited cell, and the spread rate, fire line intensity, and flame length values computed above in a burn trajectory record. Also include the terrain (e.g., 3d) distance between this cell and the ignited cell. Finally, set its **fractional-distance** value to be 0, or in the event that this bearing matches an overflow bearing from a previous iteration, set it to the **overflow-heat** value.

(i) Return a collection of burn trajectory records, one per burnable neighboring cell.

4. Main Loop

(a) If **global-clock** has not yet reached **max-runtime** and **ignited-cells** is not empty, proceed to 4.(b). Otherwise, jump to 5.(a).

(b) The timestep for this iteration of the model is calculated by dividing **cell-size** by the maximum spread rate into any cell from those cells in the **ignited-cells** map. As spread rates increase, the timesteps grow shorter and the model takes more iterations to complete. Similarly, the model has longer timesteps and takes less iterations as spread rates decrease. This is called the *method of adaptive timesteps*.

(c) If the timestep calculated in 4.(b) would cause the **global-clock** to exceed the max-runtime, then the timestep is set to the difference between **max-runtime** and **global-clock**.

(d) For each burn trajectory in **ignited-cells**:

i. Multiply the spread rate (ft/min) by the timestep (min) to get the distance traveled by the fire (ft) along this path during this iteration.

ii. Divide this distance traveled by the terrain distance between these two cells to get the new spread fraction $\in [0, 1]$ and increment the **fractional-distance** associated with the trajectory by this value.

iii. If the new **fractional-distance** is greater than or equal to 1, append this updated burn trajectory record to a list called **ignition-events**.

(e) If more than one trajectory in **ignition-events** shares the same target cell, retain only the trajectory with the largest **fractional-distance** value.

(f) For each trajectory in **ignition-events**:

i. Set the target cell's value to 1 in **fire-spread-matrix**, **flame-length** in **flame-length-matrix**, and **fire-line-intensity** in **fire-line-intensity-matrix**.

ii. If the target cell has any burnable neighbors, append an entry to **ignited-cells**, mapping this cell to each of the burn trajectories emanating from it, which are calculated by following the steps in section "Computing Burn Trajectories" above. If its **fractional-distance** value is greater than 1, add the overflow amount above 1 to the outgoing trajectory with the same bearing along which this cell was ignited. That is, if this cell was ignited by a neighbor to the southeast, then pass any overflow heat onto the trajectory leading to the northwest.

(g) Remove any trajectories from **ignited-cells** that have as their targets any of the cells in **ignition-events**.

(h) Remove any cells from **ignited-cells** that no longer have any burnable neighbors.

(i) Increment the **global-clock** by this iteration's **timestep**.

(j) Repeat from 4.(a).

5. Outputs

(a) Return an associative map with the fields shown in Table 5.

Table 5: Outputs from SIG's raster-based fire behavior model

Value	Units	Type
global-clock	minutes	double
initial-ignition-site	point represented as [row col]	vector
ignited-cells	list of points represented as [row col]	list of vectors
fire-spread-matrix	[0,1]	core.matrix 2D double array
flame-length-matrix	feet	core.matrix 2D double array
fire-line-intensity-matrix	Btu/ft/s	core.matrix 2D double array

```
(ns gridfire.fire-spread
  (:require [clojure.core.matrix :as m]
            [clojure.core.matrix.operators :as mop]
            [gridfire.fuel-models :refer [build-fuel-model moisturize]]
            [gridfire.surface-fire :refer [rothermel-surface-fire-spread-no-wind-no-slope
                                           rothermel-surface-fire-spread-max
                                           rothermel-surface-fire-spread-any
                                           anderson-flame-depth byram-fire-line-intensity
                                           byram-flame-length wind-adjustment-factor]]
            [gridfire.crown-fire :refer [van-wagner-crown-fire-initiation?
                                         cruz-crown-fire-spread
                                         crown-fire-line-intensity
                                         crown-fire-eccentricity]]))

(m/set-current-implementation :vectorz)

;; for surface fire, tau = 10 mins, t0 = 0, and t = global-clock
;; for crown fire, tau = 20 mins, t0 = time of first torch, t = global-clock
;; (defn lautenberger-spread-acceleration
;;   [equilibrium-spread-rate t0 t tau]
;;   (* equilibrium-spread-rate (- 1.0 (Math/exp (/ (- t0 t 0.2) tau)))))
;;
;; Note: Because of our use of adaptive timesteps, if the spread rate on
;;       the first timestep is not at least 83 ft/min, then the timestep will
;;       be calculated as greater than 60 minutes, which will terminate the
;;       one hour fire simulation instantly.

(defn random-cell
  "Returns a random [i j] pair with i < num-rows and j < num-cols."
  [num-rows num-cols]
  [(rand-int num-rows)
   (rand-int num-cols)])

(defn get-neighbors
  "Returns the eight points adjacent to the passed-in point."
  [[i j]]
  (let [i- (- i 1)
        i+ (+ i 1)
        j- (- j 1)
        j+ (+ j 1)]
    (vector [i- j-] [i- j] [i- j+]
            [i  j-]        [i  j+]
            [i+ j-] [i+ j] [i+ j+])))

(defn in-bounds?
  "Returns true if the point lies within the bounds [0,rows) by [0,cols)."
  [rows cols [i j]]
  (and (>= i 0)
       (>= j 0)
```

```
      (< i rows)
      (< j cols)))

(defn burnable-fuel-model?
  [^double number]
  (and (pos? number)
       (or (< number 91.0)
           (> number 99.0))))

(defn burnable?
  "Returns true if cell [i j] has not yet been ignited (but could be)."
  [fire-spread-matrix fuel-model-matrix [i j]]
  (and (zero? (m/mget fire-spread-matrix i j))
       (burnable-fuel-model? (m/mget fuel-model-matrix i j))))

(defn distance-3d
  "Returns the terrain distance between two points in feet."
  [elevation-matrix cell-size [i1 j1] [i2 j2]]
  (let [di (* cell-size (- i1 i2))
        dj (* cell-size (- j1 j2))
        dz (- (m/mget elevation-matrix i1 j1)
              (m/mget elevation-matrix i2 j2))]
    (Math/sqrt (+ (* di di) (* dj dj) (* dz dz)))))

(def offset-to-degrees
  "Returns clockwise degrees from north."
  {[-1  0]   0.0   ; N
   [-1  1]  45.0   ; NE
   [ 0  1]  90.0   ; E
   [ 1  1] 135.0   ; SE
   [ 1  0] 180.0   ; S
   [ 1 -1] 225.0   ; SW
   [ 0 -1] 270.0   ; W
   [-1 -1] 315.0}) ; NW

(defn rothermel-fast-wrapper [fuel-model-number fuel-moisture]
  (let [fuel-model      (-> (build-fuel-model (int fuel-model-number))
                            (moisturize fuel-moisture))
        spread-info-min (rothermel-surface-fire-spread-no-wind-no-slope fuel-model)]
    [fuel-model spread-info-min]))
(def rothermel-fast-wrapper (memoize rothermel-fast-wrapper))

(defn compute-burn-trajectory
  [neighbor here spread-info-min spread-info-max fuel-model crown-bulk-density
   canopy-cover canopy-height canopy-base-height foliar-moisture crown-spread-max
   crown-eccentricity landfire-layers cell-size overflow-trajectory overflow-heat]
  (let [trajectory          (mop/- neighbor here)
        spread-direction    (offset-to-degrees trajectory)
        surface-spread-rate (rothermel-surface-fire-spread-any spread-info-max
                                                               spread-direction)
```

```
        residence-time      (:residence-time spread-info-min)
        reaction-intensity  (:reaction-intensity spread-info-min)
        surface-intensity   (->> (anderson-flame-depth surface-spread-rate residence-time)
                                 (byram-fire-line-intensity reaction-intensity))
        crown-fire?         (van-wagner-crown-fire-initiation? canopy-cover
                                                               canopy-base-height
                                                               foliar-moisture
                                                               surface-intensity)
        crown-spread-rate   (if crown-fire?
                              (rothermel-surface-fire-spread-any
                               (assoc spread-info-max
                                      :max-spread-rate crown-spread-max
                                      :eccentricity crown-eccentricity)
                               spread-direction))
        crown-intensity     (if crown-fire?
                              (crown-fire-line-intensity
                               crown-spread-rate
                               crown-bulk-density
                               canopy-height
                               canopy-base-height
                               (-> fuel-model :h :dead :1hr)))
        spread-rate         (if crown-fire?
                              (max surface-spread-rate crown-spread-rate)
                              surface-spread-rate)
        fire-line-intensity (if crown-fire?
                              (+ surface-intensity crown-intensity)
                              surface-intensity)
        flame-length        (byram-flame-length fire-line-intensity)]
    {:cell                neighbor
     :trajectory          trajectory
     :terrain-distance    (distance-3d (:elevation landfire-layers) cell-size here neighbor)
     :spread-rate         spread-rate
     :fire-line-intensity fire-line-intensity
     :flame-length        flame-length
     :fractional-distance (volatile! (if (= trajectory overflow-trajectory)
                                       overflow-heat
                                       0.0))}))

(defn compute-neighborhood-fire-spread-rates!
  "Returns a vector of entries of the form {:cell [i j], :trajectory [di dj],
  :terrain-distance ft, :spread-rate ft/min, :fire-line-intensity Btu/ft/s, :flame-length ft,
  :fractional-distance [0-1]}, one for each cell adjacent to here."
  [fire-spread-matrix landfire-layers wind-speed-20ft wind-from-direction fuel-moisture
   foliar-moisture ellipse-adjustment-factor cell-size num-rows num-cols [i j :as here]
   overflow-trajectory overflow-heat]
  (let [fuel-model-number   (m/mget (:fuel-model         landfire-layers) i j)
        slope               (m/mget (:slope              landfire-layers) i j)
        aspect              (m/mget (:aspect             landfire-layers) i j)
        canopy-height       (m/mget (:canopy-height      landfire-layers) i j)
        canopy-base-height  (m/mget (:canopy-base-height landfire-layers) i j)
```

```
        crown-bulk-density  (m/mget (:crown-bulk-density landfire-layers) i j)
        canopy-cover        (m/mget (:canopy-cover       landfire-layers) i j)
        [fuel-model spread-info-min] (rothermel-fast-wrapper fuel-model-number fuel-moisture)
        midflame-wind-speed (* wind-speed-20ft 88.0
                               (wind-adjustment-factor (:delta fuel-model)
                                                       canopy-height
                                                       canopy-cover)) ; mi/hr -> ft/min
        spread-info-max     (rothermel-surface-fire-spread-max
                             spread-info-min midflame-wind-speed wind-from-direction
                             slope aspect ellipse-adjustment-factor)
        crown-spread-max    (cruz-crown-fire-spread wind-speed-20ft crown-bulk-density
                                                    (-> fuel-moisture :dead :1hr))
        crown-eccentricity  (crown-fire-eccentricity wind-speed-20ft
                                                     ellipse-adjustment-factor)]
    (into []
          (comp
           (filter #(and (in-bounds? num-rows num-cols %)
                         (burnable? fire-spread-matrix (:fuel-model landfire-layers) %)))
           (map #(compute-burn-trajectory % here spread-info-min spread-info-max fuel-model
                                          crown-bulk-density canopy-cover canopy-height
                                          canopy-base-height foliar-moisture crown-spread-max
                                          crown-eccentricity landfire-layers cell-size
                                          overflow-trajectory overflow-heat)))
          (get-neighbors here))))

(defn burnable-neighbors?
  [fire-spread-matrix fuel-model-matrix num-rows num-cols cell]
  (some #(and (in-bounds? num-rows num-cols %)
              (burnable? fire-spread-matrix fuel-model-matrix %))
        (get-neighbors cell)))

(defn select-random-ignition-site
  [fuel-model-matrix]
  (let [num-rows           (m/row-count    fuel-model-matrix)
        num-cols           (m/column-count fuel-model-matrix)
        fire-spread-matrix (m/zero-matrix num-rows num-cols)]
    (loop [[i j :as ignition-site] (random-cell num-rows num-cols)]
      (if (and (burnable-fuel-model? (m/mget fuel-model-matrix i j))
               (burnable-neighbors? fire-spread-matrix fuel-model-matrix
                                    num-rows num-cols ignition-site))
        ignition-site
        (recur (random-cell num-rows num-cols))))))

(defn identify-ignition-events
  [ignited-cells timestep]
  (->> (for [[source destinations] ignited-cells
             {:keys [cell trajectory terrain-distance spread-rate flame-length
                     fire-line-intensity fractional-distance]} destinations]
         (let [new-spread-fraction (/ (* spread-rate timestep) terrain-distance)
               new-total           (vreset! fractional-distance
```

```
                                         (+ @fractional-distance new-spread-fraction))]
          (if (>= new-total 1.0)
            {:cell cell :trajectory trajectory :fractional-distance @fractional-distance
             :flame-length flame-length :fire-line-intensity fire-line-intensity}))))
       (remove nil?)
       (group-by :cell)
       (map (fn [[cell trajectories]] (apply max-key :fractional-distance trajectories)))
       (into [])))

(defn update-ignited-cells
  [ignited-cells ignition-events fire-spread-matrix fuel-model-matrix landfire-layers
   wind-speed-20ft wind-from-direction fuel-moisture foliar-moisture ellipse-adjustment-factor
   cell-size num-rows num-cols]
  (let [newly-ignited-cells (into #{} (map :cell) ignition-events)]
    (into {}
          (concat
           (for [[cell spread-info] ignited-cells
                 :when (burnable-neighbors? fire-spread-matrix fuel-model-matrix
                                            num-rows num-cols cell)]
             [cell (remove #(contains? newly-ignited-cells (:cell %)) spread-info)])
           (for [{:keys [cell trajectory fractional-distance]} ignition-events
                 :when (burnable-neighbors? fire-spread-matrix fuel-model-matrix
                                            num-rows num-cols cell)]
             [cell (compute-neighborhood-fire-spread-rates!
                    fire-spread-matrix
                    landfire-layers
                    wind-speed-20ft
                    wind-from-direction
                    fuel-moisture
                    foliar-moisture
                    ellipse-adjustment-factor
                    cell-size
                    num-rows
                    num-cols
                    cell
                    trajectory
                    (- fractional-distance 1.0))])))))

(defn run-fire-spread
  "Runs the raster-based fire spread model with these arguments:
  - max-runtime: double (minutes)
  - cell-size: double (feet)
  - elevation-matrix: core.matrix 2D double array (feet)
  - slope-matrix: core.matrix 2D double array (vertical feet/horizontal feet)
  - aspect-matrix: core.matrix 2D double array (degrees clockwise from north)
  - fuel-model-matrix: core.matrix 2D double array (fuel model numbers 1-256)
  - canopy-height-matrix: core.matrix 2D double array (feet)
  - canopy-base-height-matrix: core.matrix 2D double array (feet)
  - crown-bulk-density-matrix: core.matrix 2D double array (lb/ft^3)
  - canopy-cover-matrix: core.matrix 2D double array (0-100)
```

```
- wind-speed-20ft: double (miles/hour)
- wind-from-direction: double (degrees clockwise from north)
- fuel-moisture: doubles (%){:dead {:1hr :10hr :100hr} :live {:herbaceous :woody}}
- foliar-moisture: double (%)
- ellipse-adjustment-factor: (< 1.0 = more circular, > 1.0 = more elliptical)
- initial-ignition-site: point represented as [row col] (randomly chosen if omitted)"
([max-runtime cell-size landfire-layers wind-speed-20ft wind-from-direction
  fuel-moisture foliar-moisture ellipse-adjustment-factor]
 (let [ignition-site (select-random-ignition-site (:fuel-model landfire-layers))]
   (run-fire-spread max-runtime cell-size landfire-layers wind-speed-20ft
                    wind-from-direction fuel-moisture foliar-moisture
                    ellipse-adjustment-factor ignition-site)))
([max-runtime cell-size landfire-layers wind-speed-20ft wind-from-direction
  fuel-moisture foliar-moisture ellipse-adjustment-factor
  [i j :as initial-ignition-site]]
 ;;(println "Fire ignited at" initial-ignition-site)
 (let [fuel-model-matrix          (:fuel-model landfire-layers)
       num-rows                   (m/row-count fuel-model-matrix)
       num-cols                   (m/column-count fuel-model-matrix)
       fire-spread-matrix         (m/zero-matrix num-rows num-cols)
       flame-length-matrix        (m/zero-matrix num-rows num-cols)
       fire-line-intensity-matrix (m/zero-matrix num-rows num-cols)]
   (when (and (in-bounds? num-rows num-cols initial-ignition-site)
              (burnable-fuel-model? (m/mget fuel-model-matrix i j))
              (burnable-neighbors? fire-spread-matrix fuel-model-matrix
                                   num-rows num-cols initial-ignition-site))
     ;; initialize the ignition site
     (m/mset! fire-spread-matrix i j 1.0)
     (m/mset! flame-length-matrix i j 1.0)
     (m/mset! fire-line-intensity-matrix i j 1.0)
     (loop [global-clock  0.0
            ignited-cells {initial-ignition-site (compute-neighborhood-fire-spread-rates!
                                                  fire-spread-matrix
                                                  landfire-layers
                                                  wind-speed-20ft
                                                  wind-from-direction
                                                  fuel-moisture
                                                  foliar-moisture
                                                  ellipse-adjustment-factor
                                                  cell-size
                                                  num-rows
                                                  num-cols
                                                  initial-ignition-site
                                                  nil
                                                  0.0)}]
       (if (and (< global-clock max-runtime)
                (seq ignited-cells))
         (let [dt              (->> ignited-cells
                                    (vals)
                                    (apply concat)
```

```
                              (map :spread-rate)
                              (reduce max 0.0)
                              (/ cell-size))
          timestep        (if (> (+ global-clock dt) max-runtime)
                            (- max-runtime global-clock)
                            dt)
          ignition-events (identify-ignition-events ignited-cells timestep)]
                          ;; [{:cell :trajectory :fractional-distance
                          ;;   :flame-length :fire-line-intensity} ...]
      (doseq [{:keys [cell flame-length fire-line-intensity]} ignition-events]
        (let [[i j] cell]
          (m/mset! fire-spread-matrix         i j 1.0)
          (m/mset! flame-length-matrix        i j flame-length)
          (m/mset! fire-line-intensity-matrix i j fire-line-intensity)))
      (recur (+ global-clock timestep)
             (update-ignited-cells ignited-cells ignition-events fire-spread-matrix
                                   fuel-model-matrix landfire-layers wind-speed-20ft
                                   wind-from-direction fuel-moisture foliar-moisture
                                   ellipse-adjustment-factor cell-size num-rows
                                   num-cols)))
    {:global-clock               global-clock
     :initial-ignition-site      initial-ignition-site
     :ignited-cells              (keys ignited-cells)
     :fire-spread-matrix         fire-spread-matrix
     :flame-length-matrix        flame-length-matrix
     :fire-line-intensity-matrix fire-line-intensity-matrix}))))))
```

This concludes our description of GridFire's raster-based fire spread algorithm.

6 User Interface

The GridFire model described in the previous section may be called directly from the REPL through the **run-fire-spread** function. However, this would require that the user had already prepared all of their map layers as 2D Clojure core.matrix values. In order to enable GridFire to easily access a wide range of raster formatted GIS layers directly, we make use of the open source Postgresql database and its PostGIS spatial extensions along with a simple client interface from the Clojure side. This interface is described in Section 6.1. Section 6.2 then describes GridFire's command line interface along with its input configuration file format.

6.1 PostGIS Bridge

Extracting raster layers from a PostGIS database is performed by a single function, called **postgis-raster-to-matrix**, which constructs a SQL query for the layer, sends it to the database in a transaction, and returns the result as a core.matrix 2D double array with no-data values represented as -1.0. The georeferencing information associated with this tile is also included in the returned results. This function may be called directly from the REPL or indirectly through GridFire's command line interface.

```
(ns gridfire.postgis-bridge
  (:require [clojure.java.jdbc :as jdbc]
            [clojure.core.matrix :as m])
  (:import (org.postgresql.jdbc4 Jdbc4Array)))

(m/set-current-implementation :vectorz)

(defn postgis-raster-to-matrix
  "Send a SQL query to the PostGIS database given by db-spec for a
  raster tile from table table-name. Optionally resample the raster to
  match resolution and set any values below threshold to 0. Return the
  post-processed raster values as a Clojure matrix using the
  core.matrix API along with all of the georeferencing information
  associated with this tile in a hash-map with the following form:
  {:srid 900916,
   :upperleftx -321043.875,
   :upperlefty -1917341.5,
   :width 486,
   :height 534,
   :scalex 2000.0,
   :scaley -2000.0,
   :skewx 0.0,
   :skewy 0.0,
   :numbands 1,
   :matrix #vectorz/matrix Large matrix with shape: [534,486]}"
  ([db-spec table-name]
   (let [meta-query (str "SELECT (ST_Metadata(rast)).* FROM " table-name)
         data-query (str "SELECT ST_DumpValues(rast,1) AS matrix FROM " table-name)]
     (jdbc/with-db-transaction [conn db-spec]
       (let [metadata (first (jdbc/query conn [meta-query]))
             matrix   (when-let [result (seq (jdbc/query conn [data-query]))]
                        (->> (first result)
                             :matrix
                             (#(.getArray ^Jdbc4Array %))
                             (m/emap #(or % -1.0))
                             m/matrix))]
         (assoc metadata :matrix matrix)))))
  ([db-spec table-name resolution threshold]
   (let [rescale-query   (if resolution
                           (format "ST_Rescale(rast,%s,-%s,'NearestNeighbor')"
                                   resolution resolution)
                           "rast")
         threshold-query (if threshold
                           (format (str "ST_MapAlgebra(%s,NULL,"
                                        "'CASE WHEN [rast.val] < %s"
                                        " THEN 0.0 ELSE [rast.val] END')")
                                   rescale-query threshold)
```

```
                         rescale-query)
        meta-query      (format "SELECT (ST_Metadata(%s)).* FROM %s"
                                threshold-query table-name)
        data-query      (format "SELECT ST_DumpValues(%s,1) AS matrix FROM %s"
                                threshold-query table-name)]
    (jdbc/with-db-transaction [conn db-spec]
      (let [metadata (first (jdbc/query conn [meta-query]))
            matrix   (when-let [result (seq (jdbc/query conn [data-query]))]
                       (->> (first result)
                            :matrix
                            (#(.getArray ^Jdbc4Array %))
                            (m/emap #(or % -1.0))
                            m/matrix))]
        (assoc metadata :matrix matrix))))))
```

6.2 Command Line Interface

The entire GridFire system is available for use directly from the Clojure REPL. This enables straightforward analysis and introspection of the fire behavior functions and their results over a range of inputs. However, if you just want to simulate an individual ignition event, GridFire comes with a simple command line interface that can be parameterized by a single configuration file, specifying the ignition location, burn duration, weather values, and the location of the PostGIS raster layers to use for topography and fuels.

The executable may be created using either the Leiningen or Boot build tools as follows:

```
lein uberjar
```

```
boot build
```

Either of these commands will generate a Java Archive (JAR) file in the **target** directory that may be run from the command line as follows:

```
java -jar gridfire-1.2.0.jar myconfig.clj
```

Note: Boot creates a JAR called “gridfire-1.2.0.jar”. Leiningen creates two files: “gridfire-1.2.0.jar” and “gridfire-1.2.0-standalone.jar”. In this case, the standalone JAR should be used, and the other JAR file may be safely ignored.

When run, the executable connects to the PostGIS database specified in the passed-in config file, downloads the necessary raster layers, simulates the ignition event for the requested duration, and returns 2D maps showing the spatial distributions of fire spread, flame length, and fire line intensity respectively. Finally, it prints out the final clock time from when the simulation was terminated as well as the total number of ignited cells on the raster grid at that point.

Which maps are created (and in what formats) may be configured by setting the following options in GridFire's input config.clj file to true or false:

1. :output-landfire-inputs?
2. :output-geotiffs?
3. :output-pngs?

```
(ns gridfire.cli
  (:gen-class)
  (:require [clojure.core.matrix :as m]
            [gridfire.postgis-bridge :refer [postgis-raster-to-matrix]]
            [gridfire.surface-fire :refer [degrees-to-radians]]
            [gridfire.fire-spread :refer [run-fire-spread]]
            [matrix-viz.core :refer [save-matrix-as-png]]
            [magellan.core :refer [register-new-crs-definitions-from-properties-file!
                                   make-envelope matrix-to-raster write-raster]]))

(m/set-current-implementation :vectorz)

(register-new-crs-definitions-from-properties-file!
 "CALFIRE" "custom_projections.properties")

(defn fetch-landfire-layers
  "Returns a map of LANDFIRE rasters with the following units:
   {:elevation          feet
    :slope              vertical feet/horizontal feet
    :aspect             degrees clockwise from north
    :fuel-model         fuel model numbers 1-256
    :canopy-height      feet
    :canopy-base-height feet
    :crown-bulk-density lb/ft^3
    :canopy-cover       % (0-100)}"
  [db-spec layer->table]
  (let [landfire-layers (reduce (fn [amap layer]
                                  (let [table (layer->table layer)]
                                    (assoc amap layer
                                           (postgis-raster-to-matrix db-spec table))))
                                {}
                                [:elevation
                                 :slope
                                 :aspect
                                 :fuel-model
                                 :canopy-height
                                 :canopy-base-height
                                 :crown-bulk-density
                                 :canopy-cover])]
    (-> landfire-layers
        (update-in [:elevation :matrix]
                   (fn [matrix] (m/emap #(* % 3.28) matrix))) ; m -> ft
        (update-in [:slope :matrix]
```

```
                   (fn [matrix] (m/emap #(Math/tan (degrees-to-radians %)) matrix)))
                   ; degrees -> %
        (update-in [:canopy-height :matrix]
                   (fn [matrix] (m/emap #(* % 3.28) matrix))) ; m -> ft
        (update-in [:canopy-base-height :matrix]
                   (fn [matrix] (m/emap #(* % 3.28) matrix))) ; m -> ft
        (update-in [:crown-bulk-density :matrix]
                   (fn [matrix] (m/emap #(* % 0.0624) matrix)))))) ; kg/m^3 -> lb/ft^3

(defn -main
  [config-file]
  (let [config              (read-string (slurp config-file))
        landfire-layers     (fetch-landfire-layers (:db-spec config)
                                                   (:landfire-layers config))
        landfire-rasters    (into {}
                                  (map (fn [[layer info]] [layer (:matrix info)]))
                                  landfire-layers)
        fire-spread-results (run-fire-spread (:max-runtime               config)
                                             (:cell-size                 config)
                                             landfire-rasters
                                             (:wind-speed-20ft           config)
                                             (:wind-from-direction       config)
                                             (:fuel-moisture             config)
                                             (:foliar-moisture           config)
                                             (:ellipse-adjustment-factor config)
                                             (:ignition-site             config))
        envelope            (let [{:keys [upperleftx upperlefty width height scalex scaley]}
                                  (landfire-layers :elevation)]
                              (make-envelope (:srid config)
                                             upperleftx
                                             (+ upperlefty (* height scaley))
                                             (* width scalex)
                                             (* -1.0 height scaley)))]
    (when (:output-landfire-inputs? config)
      (doseq [[layer info] landfire-layers]
        (-> (matrix-to-raster (name layer) (:matrix info) envelope)
            (write-raster (str (name layer) (:outfile-suffix config) ".tif")))))
    (when (:output-geotiffs? config)
      (doseq [[name layer] [["fire_spread"         :fire-spread-matrix]
                            ["flame_length"        :flame-length-matrix]
                            ["fire_line_intensity" :fire-line-intensity-matrix]]]
        (-> (matrix-to-raster name (fire-spread-results layer) envelope)
            (write-raster (str name (:outfile-suffix config) ".tif")))))
    (when (:output-pngs? config)
      (doseq [[name layer] [["fire_spread"         :fire-spread-matrix]
                            ["flame_length"        :flame-length-matrix]
                            ["fire_line_intensity" :fire-line-intensity-matrix]]]
        (save-matrix-as-png :color 4 -1.0
                            (fire-spread-results layer)
                            (str name (:outfile-suffix config) ".png"))))
```

```
    (println "Global Clock:" (:global-clock fire-spread-results))
    (println "Ignited Cells:" (count (:ignited-cells fire-spread-results)))))
```

The configuration file for GridFire's command line interface is a text file in Extensible Data Notation (EDN) format.[3] A sample configuration file is provided below and in "resources/sample_config.clj". The format should be self-evident at a glance, but it is worth noting that EDN is case-sensitive but whitespace-insensitive. Comments are anything following two semi-colons (;;). Strings are contained in double-quotes (""). Keywords are prefixed with a colon (:). Vectors are delimited with square brackets ([]). Associative lookup tables (a.k.a. maps) are delimited with curly braces ({}) and are used to express key-value relationships.

```
{:max-runtime                60.0   ;; double (minutes)
 :cell-size                  98.425 ;; double (feet)
 :wind-speed-20ft            20.0   ;; double (miles/hour)
 :wind-from-direction        90.0   ;; double (degrees clockwise from north)
 :fuel-moisture              {:dead {:1hr 0.06
                                     :10hr 0.07
                                     :100hr 0.08}
                              :live {:herbaceous 0.60
                                     :woody 0.90}} ;; doubles (%)
 :foliar-moisture            0.9 ;; double (%)
 :ellipse-adjustment-factor  1.0 ;; (< 1.0 = more circular, > 1.0 = more elliptical)
 :ignition-site              [25 40] ;; point represented as [row col]
 :db-spec                    {:classname   "org.postgresql.Driver"
                              :subprotocol "postgresql"
                              :subname     "//localhost:5432/gridfire"
                              :user        "gjohnson"}
 :landfire-layers            {:elevation          "landfire.dem WHERE rid=100"
                              :slope              "landfire.slp WHERE rid=100"
                              :aspect             "landfire.asp WHERE rid=100"
                              :fuel-model         "landfire.fbfm40 WHERE rid=100"
                              :canopy-height      "landfire.ch WHERE rid=100"
                              :canopy-base-height "landfire.cbh WHERE rid=100"
                              :crown-bulk-density "landfire.cbd WHERE rid=100"
                              :canopy-cover       "landfire.cc WHERE rid=100"}
 :srid                       "CALFIRE:900914"
 :outfile-suffix             "_run_1000"
 :output-landfire-inputs?    true
 :output-geotiffs?           true
 :output-pngs?               true}
```

This concludes our discussion of GridFire's command line interface.

[3] https://github.com/edn-format/edn

References

Frank A Albini. Estimating wildfire behavior and effects. *General technical report/Intermountain forest and range experiment station. USDA (no. INT-30)*, 1976.

Frank A Albini and Robert G Baughman. Estimating windspeeds for predicting wildland fire behavior. *USDA Forest Service Research Paper INT (USA)*, 1979.

Frank A Albini and Carolyn H Chase. *Fire containment equations for pocket calculators*, volume 268. Intermountain Forest and Range Experiment Station, Forest Service, US Dept. of Agriculture, 1980.

Hal E Anderson. Heat transfer and fire spread. *USDA forest service research paper. Intermountain and range experiment station*, (69), 1969.

Hal E Anderson. Aids to determining fuel models for estimating fire behavior. *The Bark Beetles, Fuels, and Fire Bibliography*, page 143, 1982.

Patricia L Andrews et al. Modeling wind adjustment factor and midflame wind speed for rothermel's surface fire spread model. 2012.

Robert E Burgan. Estimating live fuel moisture for the 1978 national fire danger rating system. *USDA Forest Service Research Paper INT (USA). no. 226.*, 1979.

George M Byram. Combustion of forest fuels. *Forest fire: Control and use*, 1:61–89, 1959.

Miguel G Cruz, Martin E Alexander, and Ronald H Wakimoto. Development and testing of models for predicting crown fire rate of spread in conifer forest stands. *Canadian Journal of Forest Research*, 35(7):1626–1639, 2005.

Rich Hickey. The clojure programming language. In *Proceedings of the 2008 Symposium on Dynamic Languages*, DLS '08, New York, NY, USA, 2008. ACM.

Marco Emanuel Morais. Comparing spatially explicit models of fire spread through chaparral fuels: A new model based upon the rothermel fire spread equation. Master's thesis, University of California, Santa Barbara, 2001.

Seth H Peterson, Marco E Morais, Jean M Carlson, Philip E Dennison, Dar A Roberts, Max A Moritz, David R Weise, et al. Using hfire for spatial modeling of fire in shrublands. 2009.

Seth H Peterson, Max A Moritz, Marco E Morais, Philip E Dennison, and Jean M Carlson. Modelling long-term fire regimes of southern california shrublands. *International Journal of Wildland Fire*, 20(1):1–16, 2011.

Richard C Rothermel. A mathematical model for predicting fire spread in wildland fuels. *Research paper/Intermountain forest and range experiment station. USDA (INT-115)*, 1972.

Richard C Rothermel. How to predict the spread and intensity of forest and range fires. *General technical report/Intermountain Forest and Range Experiment Station. USDA (no. INT-143)*, 1983.

Richard C Rothermel. Predicting behavior and size of crown fires in the northern rocky mountains. *Research paper/United States Department of Agriculture, Intermountain Research Station (INT-438)*, 1991.

Joe H Scott and Robert E Burgan. Standard fire behavior fuel models: a comprehensive set for use with rothermel's surface fire spread model. *The Bark Beetles, Fuels, and Fire Bibliography*, page 66, 2005.

Charles E Van Wagner. Conditions for the start and spread of crown fire. *Canadian Journal of Forest Research*, 7(1):23–34, 1977.