Departamento de Sistemas y Automática
Ingeniería en Tecnologías Industriales
Universidad Carlos III de Madrid

Modelización de un Incendio Forestal con Fast Marching

Trabajo de Fin de Grado

Autora: Carolina Nicolás Martín
Tutor: Dr. Santiago Garrido Bullón
Fecha de entrega: 21 Junio, 2017

Agradecimientos

A mis padres por todo su esfuerzo, sacrificio, paciencia, cariño y apoyo.

A mi hermana por su cariño, bondad y paciencia.

A Santiago Garrido por la oportunidad de realizar este Trabajo de Fin de Grado con él. Por su paciencia y dedicación.

A todos mis amigos del Grado, que se han acabado convirtiendo en mi pequeña familia durante estos 4 años.

Resumen

Los incendios forestales causan cada año importantes pérdidas de carácter medioambiental, económico e incluso cobran pérdidas humanas. Un modelo de predicción de incendios eficiente supone una mejora significativa en eficacia y rapidez con la que se apagarán los incendios.

En este Trabajo de Fin de Grado se pretende probar la viabilidad del algoritmo Fast Marching (FM), históricamente utilizado en aplicaciones de cálculo de trayectorias en robótica, para iniciar un proyecto de investigación de elaboración de un programa completo y relevante de predicción de incendios.

Esta viabilidad se pretende demostrar a partir de la elaboración de una aplicación de usuario en el programa Matlab que sea competitiva con otros simuladores de incendios en la actualidad. Dicha aplicación permitirá:

- Visualización del frente de propagación del fuego sobre un mapa 3D.
- Uso de mapas de terreno reales.
- Selección de puntos de inicio y fin del incendio.
- Selección de valores para el viento sobre el mapa, incluyendo magnitud y dirección.
- Visualización del frente de propagación en diferentes instantes del tiempo.

El éxito en el desarrollo de esta aplicación en Matlab supondrá la confirmación de viabilidad del algoritmo para el desarrollo de un programa de simulación de incendios forestales.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

Los incendios forestales suponen un problema muy importante en todo el mundo y más en España. Suponen cada año pérdidas muy importantes de carácter material, medioambiental, económico (llegando a alcanzar los 1776 millones de euros anuales en España, según el periódico *El Economista* [1]), en ocasiones se llega incluso a cobrarse pérdidas humanas en las labores de extinción o atrapamientos.

A continuación se presentan diferentes datos asociados a los incendios forestales en España recogidos del Área de Defensa contra Incendios Forestales (ADCIF) del Ministerio de Agricultura, Alimentación y Medio Ambiente [2], con la intención de concienciar al lector de la importancia de estos, así como sus causas y la difícil labor que resulta el extinguirlos.

1.1.1. Datos generales sobre incendios en España

La figura 1.1 contiene el gráfico que representa el número de siniestros sucedidos y superficie arbolada y no arbolada quemada por estos desde el año 1961 (6 años después de crearse Servicio de Incendios Forestales y con ello el inicio de la sistematización de los datos estadísticos referentes a los incendios forestales) hasta el año 2010. Puede apreciarse en dicho gráfico la magnitud del problema de los incendios forestales, llegando en un solo año, como 1985, a quemarse casi un 2 % de la superficie forestal total de España. Adicionalmente se puede observar que conforme han pasado los años el número de incendios no ha disminuido muy significativamente su tendencia, en cambio sí que lo ha hecho el número de hectáreas quemadas de manera bastante significativa. Esto implica una mejora en la labor de extinción de incendios por parte de los cuerpos de protección forestal conforme han pasado los años.

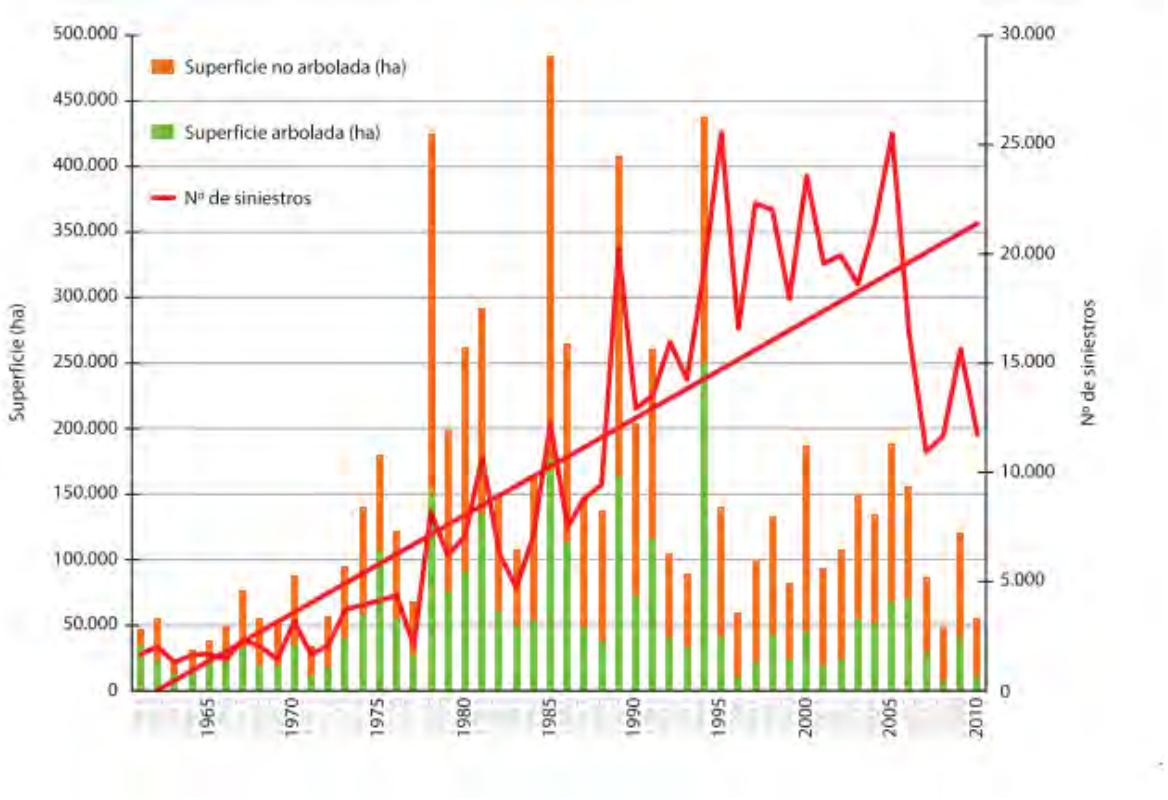


Figura 1.1: Evolución del número de siniestros y superficies afectadas, 1961-2010

Para ofrecer al lector una visión más actualizada de la situación de los incendios forestales en este país en la figura 1.2 se presenta una tabla comparando datos relativos al número de siniestros y superficie quemada en 2016 frente a la media del decenio 2006-2015.

		MEDIA DEL DECENIO 2006-2015	AÑO 2016
SINIESTROS	NÚMERO DE CONATOS (<1 ha)	8 662	6 479
	NÚMERO DE INCENDIOS (≥1 ha)	4 464	2 338
	TOTAL SINIESTROS	13 126	8 817
SUPERFICIE QUEMADA	SUPERFICIE ARBOLADA (ha)	32 102	23 174
	SUPERFICIE FORESTAL (ha)	100 958	65 817

Figura 1.2: Superficie quemada y siniestros de 2016 frente a la media del decenio 2006-2015

Es notable una reducción importante en el número de siniestros totales así como la superficie quemada en el año 2016 con respecto a la media en el decenio inmediatamente anterior. También es llamativa la reducción de incendios que hacen arder más de 1 hectárea que conatos (siniestros que queman menos de 1 hectárea de superficie). Esto implica que durante el año 2016 siniestros que antes podrían haber quemado más hectáreas actualmente queman mucho menos, lo cual se puede usar como medida de eficacia a la hora de apagar los fuegos conforme han avanzado los años. Pese a esta mejora parcial con el paso del tiempo, las hectáreas quemadas y daños ocasionados por el fuego en este último año siguen siendo muy significativos. De ahí que todavía exista mucho margen de mejora a la hora de combatir estos siniestros.

1.1.2. Causas de los incendios forestales

Las causas principales de los incendios forestales en España se pueden recoger en 5 grandes grupos:

- **Rayos**: La caída de un rayo sobre un terreno seco puede ocasionar un incendio de manera inmediata.
- **Causas intencionadas**: Causas de carácter antrópico cuyo causante era consciente del daño que hacía.

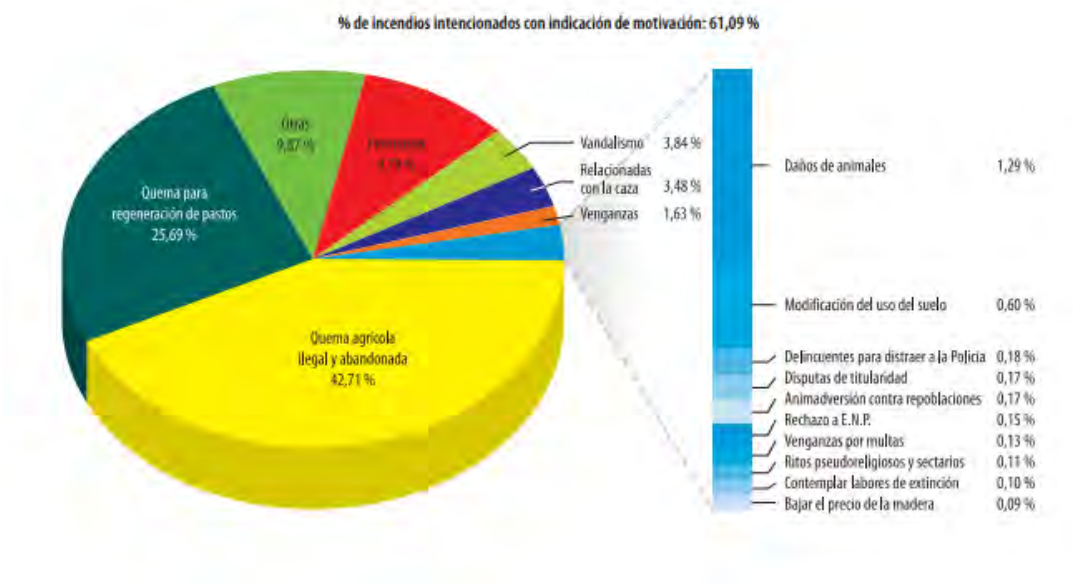


Figura 1.3: Motivaciones de incendios intencionados, 2001-2010

- **Negligencias o causas accidentales**: Causas de carácter antrópico pero no intencionadas como por ejemplo quemas agrícolas, quemas para regeneración de pastos, trabajos forestales, hogueras, fumadores, etc.
- **Reproducciones de incendios anteriores**: En ocasiones si un incendio no se sofoca de manera efectiva puede ser causante en un futuro de otros incendios.
- **Causas desconocidas**: Otras causas que no se han identificado.

En la figura 1.4 se representan la proporción de siniestros y superficie quemada por cada uno de los grupos de causas de incendio.

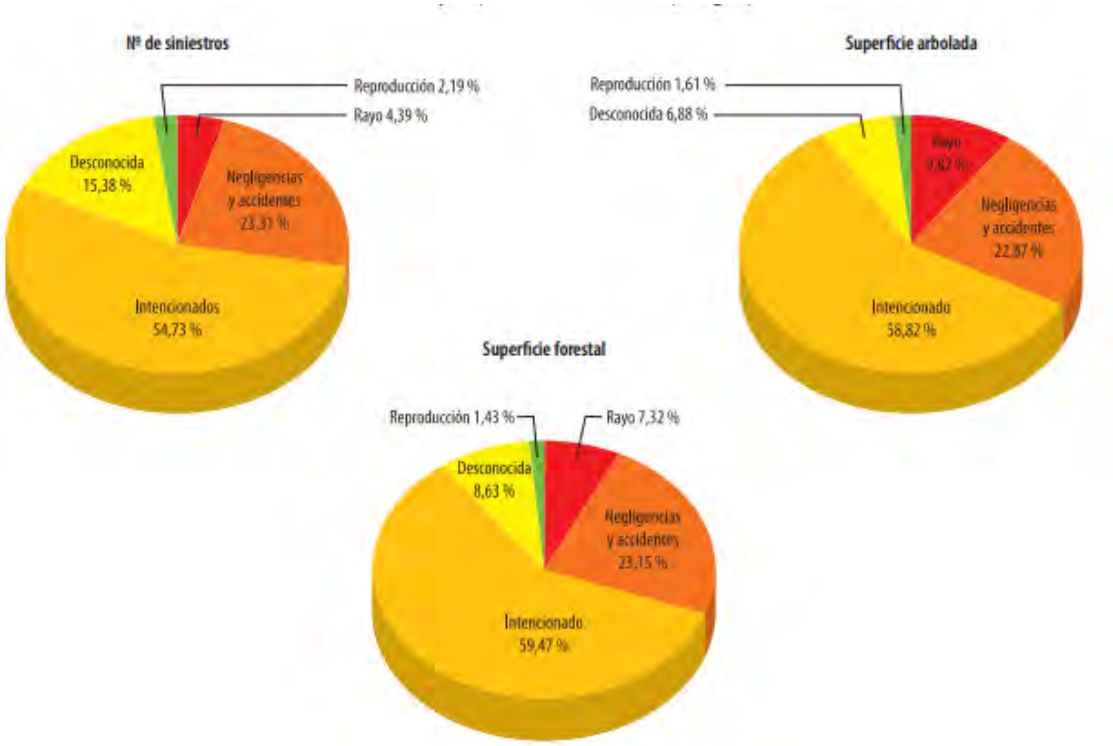


Figura 1.4: Núnmero de siniestros y superficies afectadas por grupos de causas, 2001-2010

1.1.3. Víctimas

De todas las pérdidas ocasionadas por el fuego, las pérdidas humanas son sin lugar a dudas las más importantes y la motivación más importante para combatir estos incendios.

En la figura 1.5 se presenta una tabla con información sobre víctimas mortales y heridos por causa de incendios en el decenio 2001-2010.

Año	Nº de personas fallecidas						Nº heridos entre el personal de extinción	Ajenos al personal de extinción
	Personal en labores de extinción o en labores de ida o vuelta de los incendios					Ciudadanos ajenos al personal de extinción		
	Tripulaciones	Agentes forestales y Brigadistas	Personal de Maquin./ Autobombas	Bomberos	Voluntarios			
2001	3	1	0	1	0	0	56	1
2002	3	0	1	0	0	1	60	0
2003	2	1	2	3	0	7	90	2
2004	1	2	0	0	0	2	46	0
2005	4	13	0	0	0	0	101	1
2006	0	0	1	0	0	0	58	0
2007	0	1	0	0	0	0	41	0
2008	0	0	0	0	0	1	27	0
2009	0	1	0	6	0	1	64	0
2010	2	3	0	0	0	5	11	8
TOTALES	15	22	4	10	0	17	554	12

Figura 1.5: Vctimas y heridos en incendios forestales, 2001-2010

Un dato muy relevante a tener en cuenta en cuanto a las víctimas por causa de incendios forestales es que la mayoría de los fallecimientos están ligados a atrapamientos en el fuego. Muchos de estos atrapamientos son consecuencia de la incapacidad de predecir adecuadamente la trayectoria del fuego y sus cambios en función de cambio en las condiciones del entorno (viento, cambio en la orografía, etc.).

1.1.4. Importancia de la predicción de incendios forestales

Existe una relación directa entre la mejora en la efectividad de apagado de los incendios forestales con el paso de los años y la capacidad de predicción de estos.

Es evidente que la concienciación de cada persona en contacto con los montes y la fuerte persecución y penalización de quienes intencionadamente inician fuegos son los puntos más fundamentales a la hora de reducir los incendios de una manera todavía más significativa. Pero la importancia del desarrollo de sistemas de predicción marcará la diferencia en cómo de eficiente será la extinción de los fuegos en el caso de que no se pueda evitar su comienzo. Y en la mano de estos sistemas de predicción estará también la seguridad de los brigadistas y la reducción de víctimas mortales hasta niveles mínimos o nulos debido a cambios no predichos en la trayectoria de los incendios.

Un buen sistema de predicción, que responda de manera adecuada a los cambios en los diferentes parámetros externos, permite localizar puntos claves donde habrá que combatir el fuego y cuando es más óptimo hacerlo. La capacidad de adelantarse al camino que el fuego va a tomar supone un arma decisiva de la mano de los cuerpos que se dedican a la

extinción de incendios.

A parte de la eficacia y exactitud de resultados de la predicción, un modelo de simulación de incendios será mejor cuanto menor coste computacional suponga el software desarrollado, así como la calidad de su interfaz de usuario (cuanto más amplio sea el rango de personas que lo puedan utilizar, mayor utilidad tendrá).

1.2. Objetivo

El objetivo principal de este Trabajo de Fin de Grado es comprobar la viabilidad del uso del algoritmo Fast Marching (históricamente utilizado para el cálculo de trayectorias en robots) para la elaboración de un programa de predicción de incendios forestales. Este objetivo se pretende llevar a cabo a partir del diseño de una interfaz de usuario en el programa Matlab que permita la simulación de incendios forestales sobre terrenos reales y cuya simulación dé respuesta a cambios en las condiciones externas, como el viento, puntos de inicio y fin del fuego, etc.

No se trata de la primera prueba de uso del algoritmo FM (Fast Marching) con el fin de simular incendios forestales. Adrián Bargueño en su Trabajo de Fin de Master [3] hace un primer acercamiento mediante la creación de un GUI (interfaz de usuario de Matlab) para la simulación de la propagación de un fuego en 2D para un único mapa de terreno que permite la elección de los puntos de inicio y fin de dicho fuego, y un script del mismo programa de simulación mostrando resultados en 3D.

En este TFG se pretende introducir mejoras predictivas y conseguir una una aplicación de uso sencillo para el usuario. Las aportaciones diferenciadoras son:

- Interfaz de usuario con representación en 3D .
- Uso de mapas de combustible y elevación reales que el usuario pueda elegir.
- Inclusión del viento como variable vectorial en la propagación del incendio.
- Control de los puntos de inicio y fin del incendio.
- Visualización clara de la propagación del incendio sobre el mapa.
- Contribución de la pendiente del terreno a la expansión del fuego.

Continuaría tratándose de una prueba para validar la funcionalidad de Fast Marching como algoritmo usado en una aplicación de predicción de incendios. Pero pese a esto la intención es desarrollar una interfaz lo más completa posible y que permita competir contra modelos existentes, para que, en el caso de que FM sea un algoritmo útil para este fin, confirme la viabilidad de la inversión de recursos en el desarrollo de un programa más completo que resulte de verdadera utilidad a la hora de combatir incendios.

1.3. Estructura

Este Trabajo de Fin de Grado sigue la siguiente estructura:

En el capítulo 2 se hace una introducción a los disferentes modelos de predicción de incendios existentes, así como un estudio de los programas de predicción con aplicaciones prácticas más relevantes.

En el capítulo 3 se presenta al lector los aspectos más importantes del algoritmo FM y cómo este funciona. Dado que la parte más significativa del trabajo recae en este.

En el capítulo 4 se explica el funcionamiento de la interfaz de usuario diseñada, así como los aspectos más relevantes de su programación.

En el capítulo 5 se presentan los resultados proporcionados por la interfaz de usuario para diferentes casos de incendio y se presentan un análisis crítico y conclusiones sobre el trabajo realizado.

En el capítulo 6 se proponen aspectos donde mejorar el trabajo realizado y pautas sobre cómo hacerlo.

En el Apéndice A se hace referencia al marco regulador del proyecto.

En el Apéndice B se hace referencia al impacto socio-económico del proyecto así como su presupuesto.

En el Apéndice C se presentan los códigos más relevantes del trabajo.

Capítulo 2

Estado del Arte

Se ha probado la importancia de un programa fiable de predicción de incendios forestales a la hora de reducirlos y minimizar todo lo posible los daños ocasionados por estos. Por esta razón en los últimos años varios han sido los intentos de conseguir programas de predicción fiables, económicos y con mínimo coste computacional y mínima demanda de datos externos para realizar una predicción fiable.

La esencia de un programa de predicción de incendios reside en el modelo utilizando para realizar la predicción de la trayectoria a recorrer por el fuego. Al tratarse de un fenómeno tan complejo, como lo es la trayectoria del fuego en un incendio, existen diferentes lineas de investigación en lo que respecta a modelos de predicción. Teniendo en cuenta que la propagación del fuego depende de numerosísimos factores (flamabilidad del terreno, viento, humedad, pendiente del terreno, etc.) es difícil dar con un único enfoque de modelo que pueda tener en cuenta todas estas variables y predecir de manera acorde.

A continuación se describen los tipos de modelos utilizados hasta la fecha, en qué consiste cada uno, así como las líneas de trabajo más significativas entre los mismos.

2.1. Modelos físicos

Son modelos que hacen uso de leyes físicas y químicas relativas al propagación del fuego en función de variables que lo propician. Abarcan desde modelos que se basan en leyes generales de transferencia de masa y calor hasta modelos computacionales basados en dinámica de fluidos.

Se podrían dividir las variables de las que depende la manera en la que se propaga un incendio en tres grandes bloques:

- **Meteorología**: A continuación ejemplos de diferentes aspectos meteorológicos y como pueden afectar :
  - Viento, crucial en la velocidad de propagación del fuego, así como en cambios en la trayectoria de este.
  - Temperatura, sea del terreno o del aire, muy relevante en cualquier modelo de trasferencia de calor o dada su importancia en la dinámica de las corrientes de aire.
  - Humedad, la concentración de agua en el ambiente tiene su efecto físico sobre la facilidad de trasmisión del fuego.
- **Topografía**: Temas como la rugosidad o la pendiente del terreno pueden tener efecto físico sobre la trayectoria del fuego de manera directa, o tener incluso efecto (en el caso de la pendiente) indirecto, ya que terrenos desiguales y montañosos el viento se comporta de manera diferente que en terrenos planos.

- **Combustible**: Se trata del único bloque sobre el cual se podría actuar para prevenir o extinguir un incendio. Características como la humedad del terreno, la cantidad de combustible que exista, el tipo de vegetación o la disposición de esta tienen una serie de relaciones físicas con la intensidad de propagación o el tipo de propagación del incendio. Esas relaciones físicas pueden desarrollarse para predecir el efecto que pueden tener sobre el fuego y a partir del conocimiento de ese efecto se podría incluso modificar dicho combustible antes o durante un incendio.

Cada una de las variables mencionadas anteriormente afecta de manera diferente al fuego, y por tanto existen modelos que tratan su influencia de manera individual:

Por ejemplo, en [4] se estudia la relación entre cambios en la propagación del fuego debido a los cambios de flujo en el entorno. Estos cambios en el flujo se deben a dos factores simultáneos: por una parte el flujo de aire inducido por el propio fuego y por otra el inducido por corrientes de viento externas.

Un programa de simulación forestal completo basado en un modelado físico debería reunir varios modelos que relacionen la propagación del fuego con diferentes variables. El principal problema de los modelos físicos, sin embargo, es el constante pulso entre la exactitud y la economía computacional de estos. En numerosas ocasiones el modelado del incendio se hace por medio de ecuaciones diferenciales parciales que representan procesos térmicos o de propagación de fluidos. Cuantos más parámetros se deseen tener en cuenta y cuanta mayor sea la exactitud de la solución de dichas ecuaciones mayor será el coste computacional del modelo, y por tanto el tiempo de simulación y coste económico.

2.2. Modelos empíricos

Son modelos que concluyen relaciones entre parámetros que afectan al fuego basándose en resultados estadísticos experimentales (bien sea sobre incendios reales ocurridos o ensayos controlados) . Los resultados de estos modelos son obtenidos de manera meramente experimental y no se incorpora ningún estudio de la relación física entre las variables estudiadas y la propagación. Aún sin tener en cuenta las relaciones físicas entre variables estos modelos empíricos pueden establecer la influencia en el fuego de fenómenos como el viento, la temperatura, la humedad del combustible, etc. Dicha influencia puede estudiarse, en muchos casos, mediante relaciones físicas, pero es posible que por la dificultad bien de modelaje o de computación del modelo físico sea más adecuado una aproximación experimental.

Los modelos empíricos se pueden dividir en dos tipos:

- **Experimentos realizados en campo**: Se trata de quemas controladas en una determinada extensión de terreno. Variables como el combustible o la humedad del

terreno son normalmente mantenidos constantes mientras que se estudian casos en los que se varían condiciones de viento o de posición y número de focos de origen. El inconveniente más importante de este tipo de ensayos es la restricción en el uso de los datos estadísticos obtenidos como resultado en los modelos a los casos en los que se cumplen las condiciones que se han dado en los experimentos. Un ejemplo de este tipo de estudios es [5].

- **Experimentos realizados en laboratorio**: Se trata de recreaciones en laboratorio de situaciones de un incendio. El rango de condiciones a variar es más amplio que en el caso de los experimentos de campo: se pueden variar los combustibles, humedad, temperatura, modificar condiciones del viento, etc. Sin embargo al tratarse de situaciones de laboratorio ideales la fiabilidad de estos estudios es menor que la de aquellos experimentos en campo, y por tanto para su validación en ocasiones se necesitan contrastar los datos obtenidos. Pese a este último inconveniente y a la limitación de escala en los ensayos, resultan más económicos, sencillos y seguros que los realizados en campo. Un ejemplo de este tipo de estudios es [6].

A diferencia de los modelos físicos el coste computacional de los modelos empíricos es muy baja y pueden usarse de manera inmediata al tratarse de datos almacenados. Sin embargo sí existe un desembolso muy significativo (menor en el caso de los experimentos de laboratorio) a la hora de realizar los experimentos para obtener los datos del modelo.

2.3. Modelos computacionales

En los últimos años se han desarrollado varios modelos computacionales. Se trata de modelos que utilizan la discretización de alguna ecuación de propapagación y aplican un algoritmo de resolución sobre una malla discreta.

Uno de los modelos más importantes dentro de los de este tipo es [7]. Este modelo asume que el fuego se propaga en una malla cuyas celdas están constituidos por combustibles homogéneos y la disposición de tipos de combustibles dentro de una malla para un caso concreto se determina por distribución probabilística y el procedimiento de muestreo de Monte Carlo. Tanto la velocidad como la forma del incendio está determinada por la humedad y tipo de combustible en cada celda, esto se consigue mediante el cálculo del tiempo mínimo necesario para que el fuego alcance la siguiente celda por medio de un algoritmo de programación dinámico. Adicionalmente este modelo tiene en cuenta el tiempo de persistencia, definido como tiempo en el que continúa ardiendo el combustible una vez que el frente del incendio ha sobrepasado la zona de dicho combustible. También considera la posibilidad de autoapagado, cuando el fuego se encuentra rodeado de un combustible muy húmedo, por ejemplo. La principal desventaja de este modelo es que no tiene en absoluto en cuenta el efecto del viento o de la pendiente del terreno.

2.3.1. Uso de modelos usando métodos de *Level Set* y Fast Marching

De los estudios realizados sobre la propagación del fuego se ha llegado a la conclusión de que el camino que el fuego toma para ir de un punto a otro es aquel que minimiza el tiempo que tarda, el cual no tiene porqué coincidir con el camino que minimiza la distancia física. A partir de este este hecho se presenta la posibilidad de calcular la trayectoria del frente del fuego mediante el método de conjunto de nivel, del cual Fast Marching es un caso particular. Se presenta una explicación extensa en el capítulo 3.

En [8] se comienza a estudiar la posibilidad de incluir esta metodología en la simulación de incendios y compararla para probar su validez en el campo. [9] constituye un modelo más completo de análasis usando métodos de nivel. Estudia el efecto en la forma y velocidad de propagación del perímetro del frente del fuego que tienen tres parámetros: las variaciones en combustión, las variaciones en la dirección del viento y las variaciones en la velocidad del mismo.

En España el trabajo más significativo en este campo (predicción de incendios mediante Fast Marching) ha sido el desarrollado por Adrián Bargueño [3], Santiago Garrido y Luis Moreno, cuya línea de trabajo se continúa en este Trabajo de Fin de Grado.

2.4. Modelos mixtos

Vistas las ventajas e inconvenientes de cada uno de los métodos señalados anteriormente, no es difícil plantearse que la utilización de modelos que combinen resultados empíricos con modelos físicos o computacionales pueden ser muy beneficiosos a la hora de buscar resultados completos, exactos y eficientes a la hora de predecir la trayectoria del fuego. Adicionalmente, los modelos mixtos pueden servir de comparación en sí mismos, ya que en ocasiones se realiza un estudio paralelo sobre la influencia de una variable en la trayectoria del fuego de manera experimental y de manera desarrollada física o matemáticamente, y de esta manera, contar con información más fiable.

Un conocido e importante modelo de este tipo es [10]. Consiste en un muy completo modelo que estudia complementando modelos de predicción matemáticos basados en leyes físicas con experimentos en túneles de viento. Este modelo tiene en cuenta la influencia características del combustible (como la profundidad, humedad de o la relación superficie-volumen de sus partículas), variables de carácter meteorológico como la velocidad media del viento o aquellos relacionados con la orografía, como la pendiente del terreno.

2.5. Programas de predicción más relevantes

Se ha hecho un breve análisis de los diferentes métodos estudiados para ver la influencia de diferentes variables del entorno en la expansión de un incendio forestal. Sin embargo, si se pretende la elaboración de una herramienta que sirva a usuarios externos para la visualización de la trayectoria que vaya a tomar un incendio es preciso tener en cuenta muchos aspectos. A parte del núcleo del programa que será la elección del modelo (o modelos, si se quieren tener en cuenta muchas variables), se necesita una interfaz de usuario adecuada, recopilación de datos sobre variables externas y un software adecuado. Por esta razón, la elaboración de un interfaz útil de predicción de incendios requiere un inmenso trabajo y la colaboración de programadores, modeladores, especialistas en el sector, etc.

A continuación se hace una breve descripción de varios de los programas más relevantes en la simulación de incendios en la actualidad:

2.5.1. FIRESTAR

Se trata de un proyecto de asistencia sobre riesgos y eficacia preventiva de incendios a diferentes usuarios dentro de la zona del Mediterráneo.

Su principal característica es que utiliza esencialmente un modelo de predicción de carácter puramente físico [11]. Este se basa en la resolución de una serie de ecuaciones diferenciales parciales que se basan en los principios de conservación de masa, energía o momento. El modelo global conseguido predice el efecto que tendrán numerosas variables de interés (temperatura, densidad, velocidad del gas, etc.) entre ellas mismas y sobre el incendio.

Como ejemplo se presenta en la figura 2.1 la influencia que tienen diferentes condiciones de viento (U_h) sobre campos como la temperatura (representado con diagrama de colores) y la velocidad del gas (representado como campo vectorial).

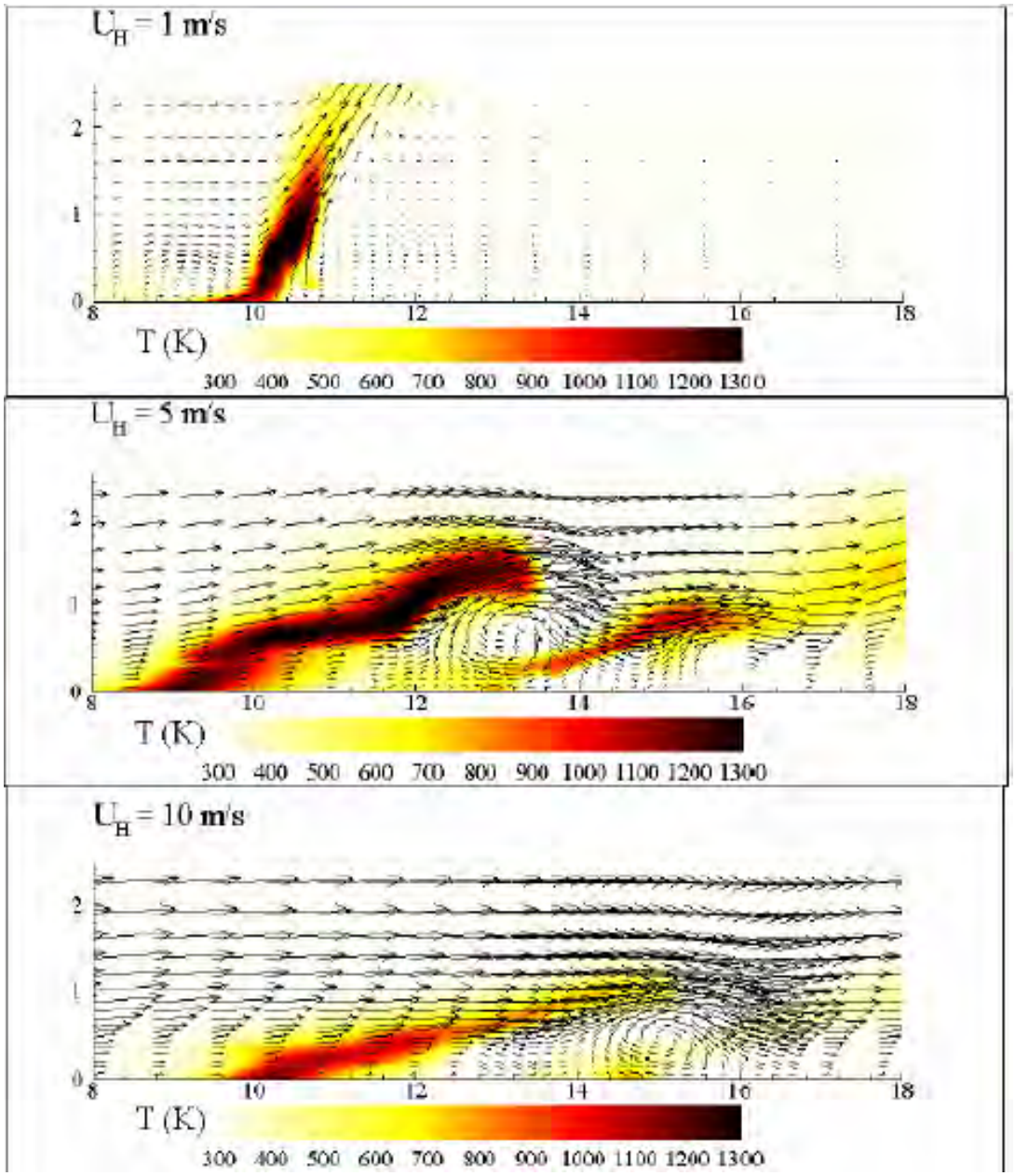


Figura 2.1: Campos de temperatura de gas y velocidad de gas obtenidos a 3 velocidades de viento diferentes

El principal inconveniente del modelo utilizado en este proyecto es el hecho de que únicamente puede trabajar en 1 y 2 dimensiones (debido a la dificultad computacional de desarrollar un programa basado en un modelo puramente físico en 3 dimensiones).

2.5.2. FireRS

Acrónimo de wildFIRE Remote Sensings [12] , es un proyecto en desarrollo todavía, pero se considera importante su mención por el impacto que tendrá en los próximos años.

Su iniciativa es proporcionar a las agencias de emergencia y centros de coordinación de incendios forestales una herramienta completa e innovadora que incluye información en tiempo prácticamente real, perímetro del fuego, imágenes infrarrojo, posicionado en GPS, protocolos de actuación y, lo más importante, predicción de la propagación entre otras herramientas.

Es un proyecto con un presupuesto de casi 2 millones de euros que cuenta con sensores infrarrojos, una flota de UAVs autónomos y un picosatélite dedicado a la comunicación y un centro de control que gestionará información involucrada. Comenzó el 01/07/2016 y tiene prevista su finalización el 30/06/2019. Está mayormente financiado por el Fondo Europeo de Desarrollo Regional (FEDER) y colaboran la Universidad de Vigo (Coordinador), la Universidad de Oporto y el Centre National de la Recherche Scientifique.

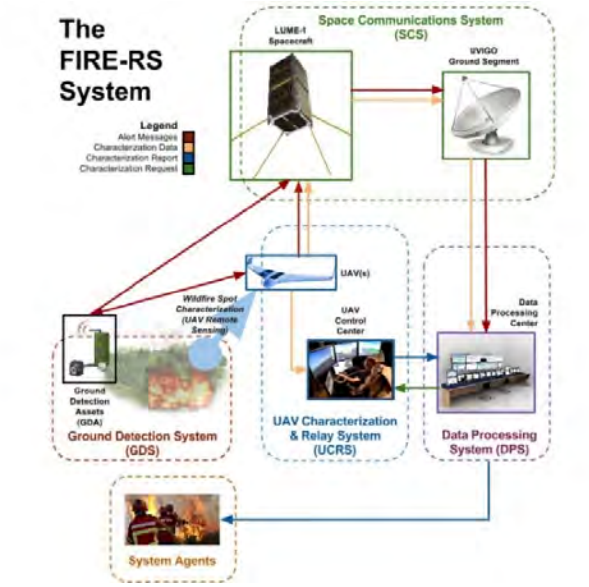


Figura 2.2: Diagrama del proyecto FireRS

2.5.3. Prometheus

Prometheus [13] es un modelo determinista (los resultados de la simulación pueden ser directamente comparados con los *inputs* de esta) de simulación de propagación de

incedios forestales. Y es el método de predicción más importante usado en las zonas forestales canadienses.

Utiliza variables de carácter topográfico como la pendiente o elevación, diferentes tipos de combustible y diferentes modelos meteorológicos para la simulación del fuego utilizando un modelo computacional basado en el principio de Huygens. Este principio se basa en un descubrimiento: la propagación de ondas de luz de que en cada momento, el frente de onda contiene los centros de nuevas pequeñas propagaciones individuales cuya envolvente tangencial corresponde al frente de onda en el siguiente instante. Este fenómeno se puede observar gráficamente en la figura 2.3.

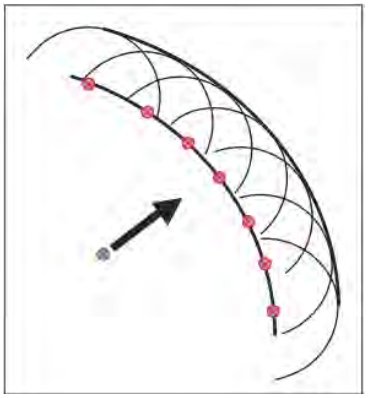


Figura 2.3: Formación de un nuevo frente de onda usando el principio de Huygens

La estrategia para simular la propagación del fuego a partir de este principio es asumir que el frente de fuego puede considerarse un polígono y que en cada vértice propaga una onda independiente en función de las características de cada vértice (combustible, topografía o meteorología). El siguiente frente de onda del fuego será la envolvente de estas ondas independientes en los vértices. El proceso se ilustra en la figura 2.4.

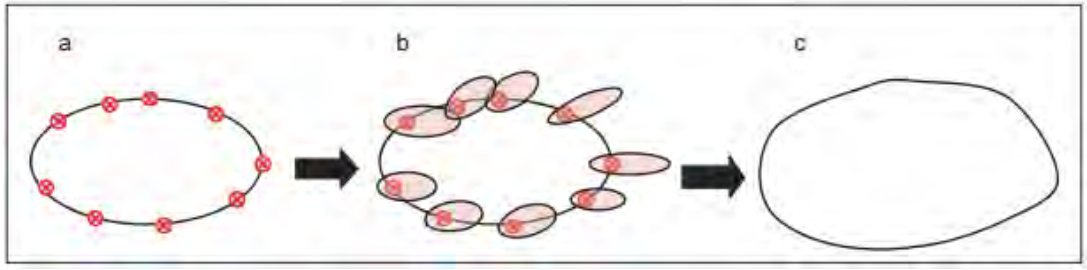


Figura 2.4: Aplicación del principio de Huygens a la propagación del fuego

Una parte esencial de cualquier programa de simulación de incendios es el software utilizado para la implementación del modelo. Prometheus utiliza COM (Common Object Model), una herramienta de programación de Microsoft que permite la utilización de

componentes orientados a objetos que pueden ser exportados a otros entornos de programación sin compartir el código original. Contiene una COM de mayor nivel (PrometheusCOM) y 5 de menor nivel (FuelCOM, FWICOM, GridCOM, FireEngine, WeatherCOM) encargados de las variables externas de las que depende el modelo.

Existen aplicaciones software con interfaz de usuario de simulación de incendios como Burn-P3 [14] que utiliza Prometheus como núcleo de programación. Burn-P3 utiliza como *inputs* información en formato ASCII sobre información de combustión (único *input* obligatorio para que funcione), elevación del terreno, meteorología, regiones del mapa para las que el fuego varía y mapas de efecto de la topografía sobre la expansión del fuego. La representación de los mapas con dichos *inputs* se pueden apreciar en la figura 2.5.

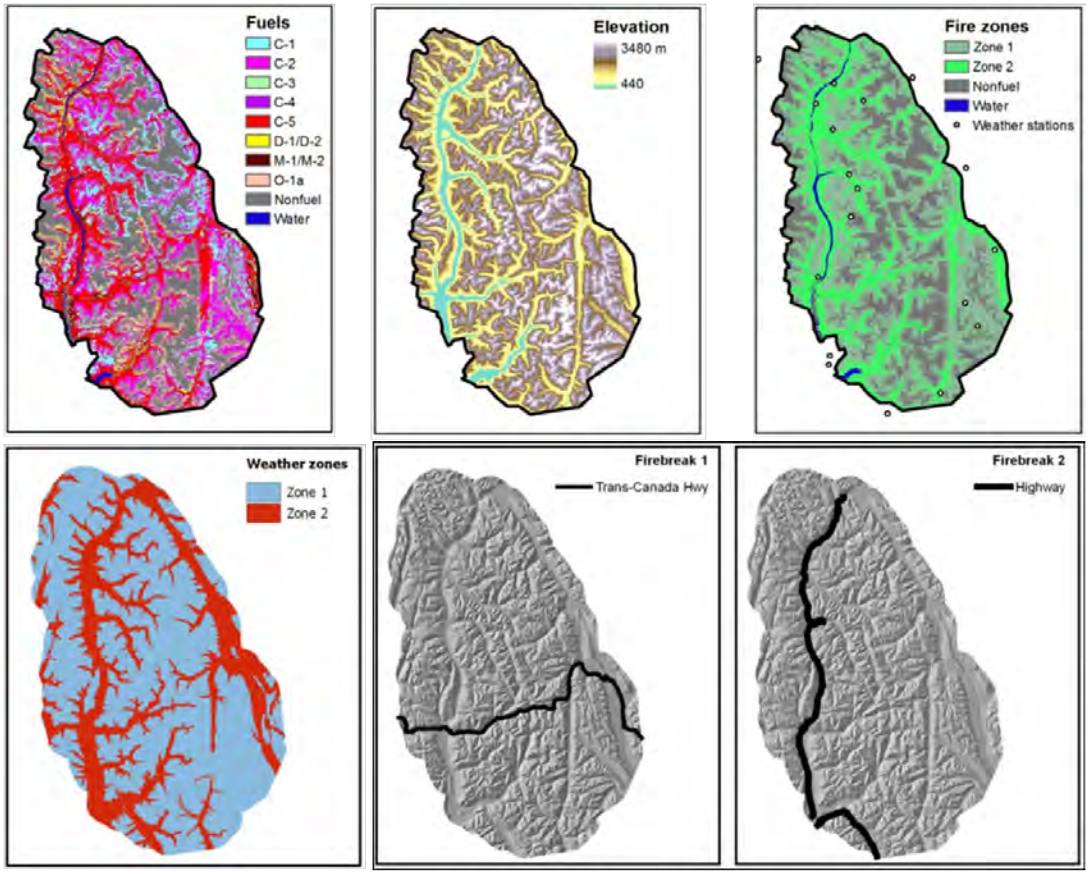


Figura 2.5: Representación de los mapas de *inputs* en Burn-P3

Burn-P3 produce 3 tipos de *output*: un mapa con información sobre la probabilidad de que una zona se queme (figura 2.6), una tabla estadística sobre el mismo y un archivo tipo log.

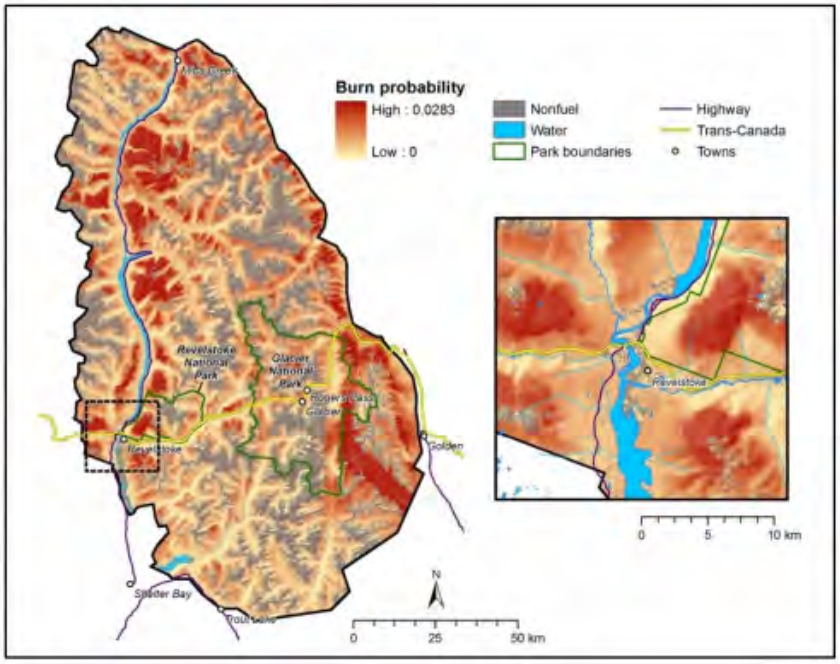


Figura 2.6: Mapa de probabilidad de que el terreno se queme

2.5.4. FARSITE

Se trata probablemente del simulador de incendios forestales más completo que existe hasta la fecha.

FARSITE [15] utiliza como modelo de predicción el principio de Huygens (explicado en la sección 2.5.3) en combinación con modelos existentes como el modelo de fuego de superficie de Rothermel o la iniciación de fuego de copas de Van Wagner.

El procedimiento de uso se muestra en el diagrama de la figura 2.7. Las acciones a realizar por el usuario sobre el programa se muestran como rectángulos y los *inputs* necesarios (archivos sobre mapas de elevación, fuel, meteorología, etc.) necesarios o opcionales en cada punto del programa, así como los tipos de archivos que se pueden extraer del programa en cada punto se muestran como carpetas.

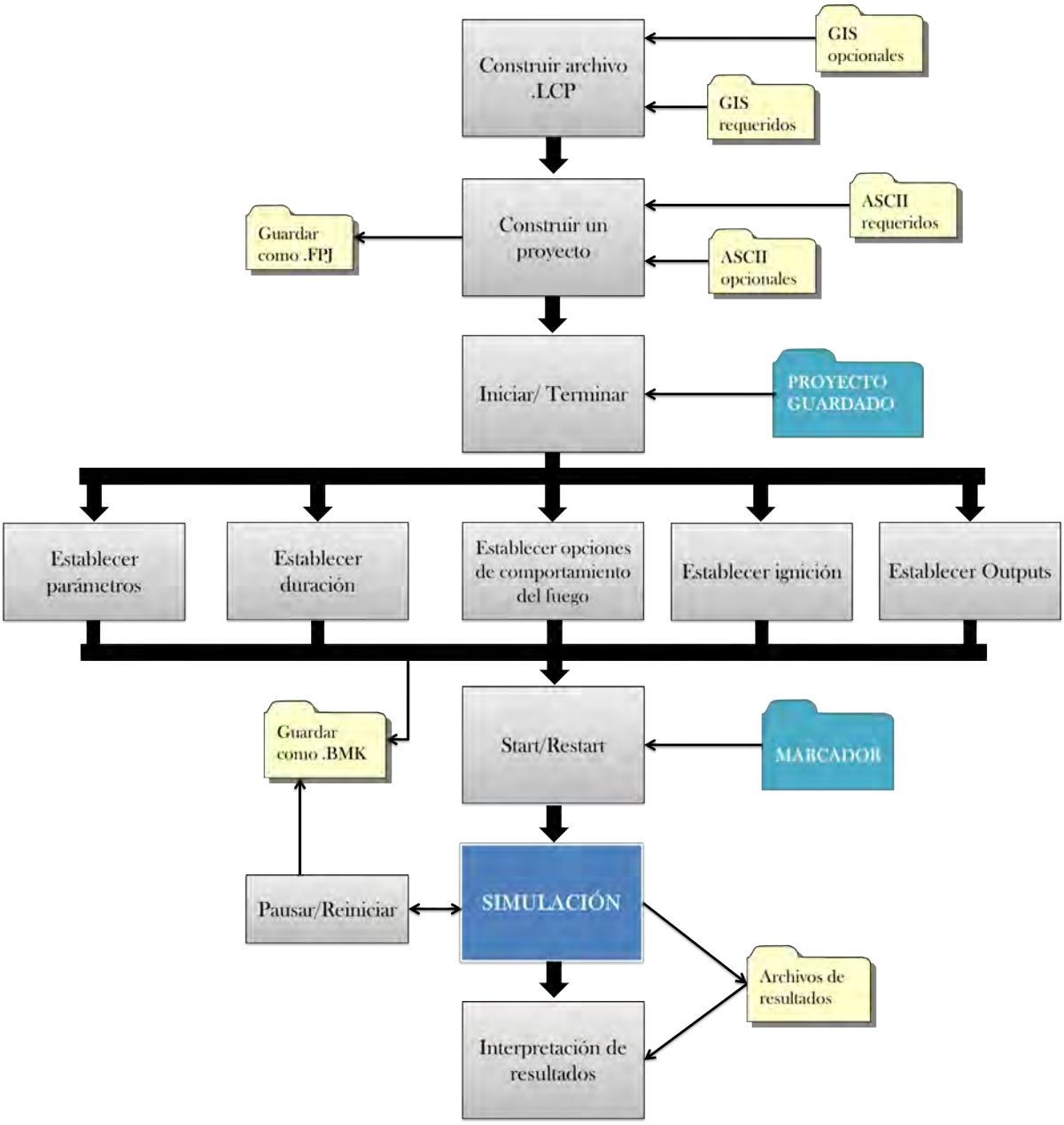


Figura 2.7: Diagrama explicativo del funcionamiento de FARSITE

En la figura 2.8 se muestra la representación gráfica de los resultados de una simulación en FARSITE [16]. Se trata de un programa muy completo, que posibilita incluir mucha información sobre variables que afectan al fuego y permite también conocer muchos datos sobre la propagación del fuego. Sin embargo tiene un inconveniente relativo a la representación de resultados. En la zona rodeada en rojo de la figura 2.8 se encuentran las líneas que representan el desarrollo del incendio con el transcurso del tiempo. Estas líneas son poco nítidas, no se asemejan lo suficiente al estándar de representación de un frente de propagación y cuesta mucho diferenciarlas del resto del terreno (de colores más

vivos). Adicionalmente el programa no facilita la variación de los valores de las distintas variables de una manera fácil y directa como sería de esperar en una interfaz de usuario.

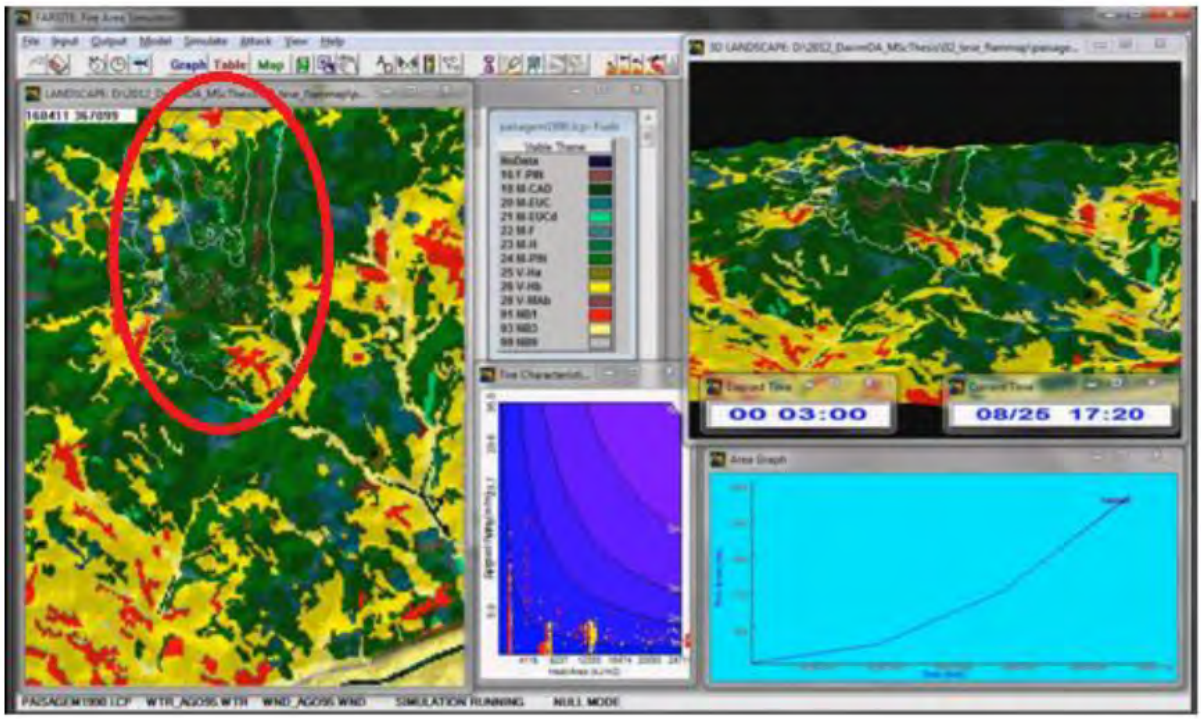


Figura 2.8: Interfaz de FARSITE mostrando resultados de simulación

Capítulo 3

Metodología

Como ha sido determinado en la sección 1.2, el objetivo principal de este trabajo es el desarrollo de una interfaz de usuario en Matlab mediante la implementación del algoritmo Fast Marching (históricamente orientado a aplicaciones de robótica y visión digital) que permita la predicción de incendios forestales.

El método de Fast Marching es un caso específico del método de *Level Set*, inicialmente propuesto para simular la propagación de una onda en un espacio discretizado. Fue desarrollado por Osher S. y Sethian J.A. [17]. Su utilidad radica en la extremada eficiencia con la cual resuelve la ecuación de propagación de onda Eikonal.

En este capítulo se trata de explicar al lector el funcionamiento del algoritmo FM y porqué se presenta como un método adecuado para la predicción de la trayectoria del fuego. En 3.1 se pretende dar una visión intuitiva del método y cómo este se puede aplicar a la predicción de incendios. En 3.2 se introduce la ecuación Eikonal y se explica su resolución discretizada basada en [18]. En 3.3 y 3.4 se explica el funcionamiento del algoritmo FM a partir del algoritmo de Dijkstra con la ayuda de dos ejemplos. En 3.5 se explica la influencia de un campo vectorial externo sobre el algoritmo FM.

3.1. Visión intuitiva del método Fast Marching

Una de las maneras más gráficas de entender el algoritmo FM es pensar en la expansión de una onda causada al lanzar una piedra a un recipiente con un fluido. Es sencillo entender que tanto la forma que tomará la onda, como el camino que recorrerá y el tiempo que le llevará llegar desde el punto de impacto a otro determinado punto dentro del recipiente dependerá de qué fluido o fluidos el recipiente contenga. De manera que si, por ejemplo, el recipiente estuviese lleno únicamente de agua la onda de propagación será circular, mientras que si se mezclara agua con otro fluido de diferente viscosidad como el aceite, el frente de propagación adoptaría otra forma. De la misma manera, el tiempo y el camino que la onda recorre hasta llegar a un punto determinado variarán en función del fluido en cuestión.

Por tanto, y con la intención de generalizar lo visto en el ejemplo, se ve que la función tiempo de la onda (expansión de esta) depende de la composición de las viscosidades del fluido, es decir depende del campo de velocidades sobre el que la onda se propague. El trayecto que la onda toma se puede calcular mediante descenso de gradiente (siguiendo el cambio más abrupto de este).

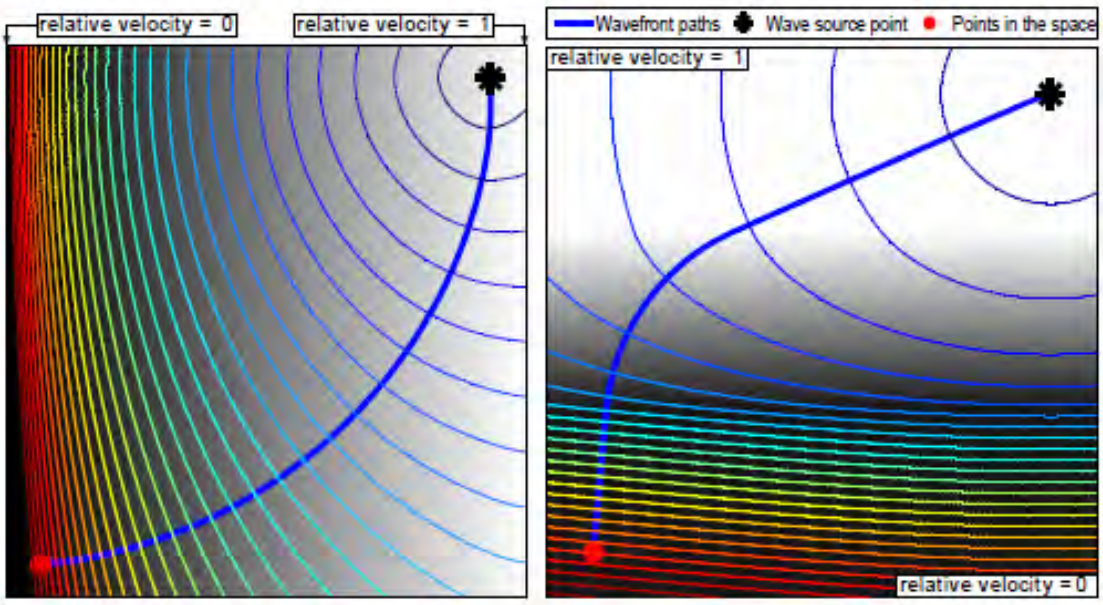


Figura 3.1: Ejemplos de propagación de una onda en medios con diferentes campos de velocidades

En el caso particular de un incendio forestal, el fuego equivale a la onda de propagación y nuestro campo de velocidades viene dado por el terreno que arde (zonas secas arderán más que las húmedas, etc.).

Una asunción importante del método FM es que el campo de velocidades es siempre positivo. Esto se traduce en que en ningún momento el frente de onda retrocederá. En prácticamente todos los casos físicos está condición es fácilmente respetable, y el caso de un fuego propagándose no es excepción (si ignoramos el efecto del viento o la acción de un combustible especialmente húmedo).

3.2. Resolución discreta de la ecuación Eikonal

3.2.1. Presentación de la ecuación Eikonal

Se tiene un dominio $\chi \subset \mathbb{R}^N$ y una función de velocidades de dicho dominio $F_{(x)}$, la cual representa la velocidad local que la onda puede adquirir en cada momento. Si se quiere llevar dicha onda desde un set de inicio $\chi_s \subset \chi$ a un set final $\chi_{final} \subset \chi$ de la manera más rápida posible, la ecuación Eikonal determina el tiempo de llegada a cada punto del dominio, $T_{(x)}$ de la siguiente manera:

$$\left|\nabla T_{(\mathbf{x})}\right| F_{(\mathbf{x})} = 1 \tag{3.1}$$

Donde para $\chi_s \subset \chi$:

$$T_{(\mathbf{x})} = 0 \tag{3.2}$$

Para un mejor entendimiento de la ecuación, es útil pensar en el caso unidimensional de la ecuación. En este caso el gradiente $\left|\nabla T_{(x)}\right|$ se puede denotar como:

$$|\nabla T| = \frac{dT}{d\theta} \tag{3.3}$$

Donde θ representa la distancia. De esta manera es intuitivo ver que de la expresión que relaciona la distancia con velocidad y tiempo:

$$\theta = FT \tag{3.4}$$

Y por tanto, derivando con respecto a θ en ambos lados de la ecuación y utilizando la igualdad (3.3) se obtiene la ecuacin Eikonal en una dimensión:

$$1 = F\,|\nabla T| \tag{3.5}$$

3.2.2. Discretización de la ecuación Eikonal

Se puede asumir que el dominio sobre el que se trabaja es un hipercubo de N dimensiones, representado con una malla en coordenadas cartesianas que contiene las funciones discretizadas F y T.

Para una mejor compresión se tratará el caso más práctico en el que $N = 2$, es decir, el caso bidimensional. En este caso tanto la función F como T se discretizan como un mapa de celdas donde i denota las filas y j las columnas de manera que $T_{i,j}$ y $F_{i,j}$ serán los valores de las celdas en el punto (i, j) de los mapas de la función tiempo y velocidad respectivamente.

A continuación se presenta la discretización de primer orden la ecuación Eikonal más común desarrollada por Osher S. y Sethian J.A [17].

$$\left\{ \begin{array}{l} max({D_{ij}}^{-x}T, 0)^2 + min({D_{ij}}^{+x}T, 0)^2 \\ max({D_{ij}}^{-y}T, 0)^2 + min({D_{ij}}^{+y}T, 0)^2 \end{array} \right\} = \frac{1}{{F_{i,j}}^2} \tag{3.6}$$

En esta discretización se usa un *upwind difference scheme* (parte de métodos de discretización) para aproximar las derivadas parciales de T. Tal que:

$$\begin{array}{l} \frac{\partial T}{\partial x} \approx {D_{ij}}^{\pm x}T = \frac{T_{i+1,j} - T_{i,j}}{\pm \triangle x} \\ \frac{\partial T}{\partial y} \approx {D_{ij}}^{\pm y}T = \frac{T_{i,j+1} - T_{i,j}}{\pm \triangle y} \end{array} \tag{3.7}$$

Donde ${D_{ij}}^{\pm x}$ representa la derivada parcial numérica lateral en la dirección $\pm x$ y, $\triangle x$ y $\triangle y$ el espaciado de la malla en las direcciones x e y respectivamente.

Otra solución más simple, aunque menos precisa es:

$$\left\{ \begin{array}{l} max({D_{ij}}^{-x}T, -{D_{ij}}^{+x}T, 0)^2 \\ max({D_{ij}}^{-y}T, -{D_{ij}}^{+y}T, 0)^2 \end{array} \right\} = \frac{1}{{F_{i,j}}^2} \tag{3.8}$$

Si sustituimos (3.7) en (3.8) y se toma:

$$\begin{array}{c} T = T_{i,j} \\ T_x = min(T_{i+1,j}, T_{i-1,j}) \\ T_y = min(T_{i,j+1}, T_{i,j-1}) \end{array} \tag{3.9}$$

Se puede reescribir la ecuación Eikonal para 2D como:

$$max\left(\frac{T - T_x}{\triangle x}, 0\right)^2 + max\left(\frac{T - T_y}{\triangle y}, 0\right)^2 = \frac{1}{{F_{i,j}}^2} \tag{3.10}$$

Cabe destacar que, dado que se ha asumido que el frente de velocidades es siempre positivo, T será siempre mayor que T_x y T_y siempre que el frente de onda no haya llegado al punto (i, j), por tanto se puede simplificar (3.10):

$$\left(\frac{T - T_x}{\triangle x}\right)^2 + \left(\frac{T - T_y}{\triangle y}\right)^2 = \frac{1}{{F_{i,j}}^2} \tag{3.11}$$

3.2.3. Solución de la ecuación Eikonal discretizada

La ecuación (3.11) puede ser representada como una ecuación de segundo grado de la forma $aT^2 + bT + c = 0$ donde:

$$\begin{array}{c} a = \triangle x^2 + \triangle y^2 \\ b = -2(\triangle y^2 T_x + \triangle x^2 T_y) \\ c = \triangle y^2 {T_x}^2 + \triangle x^2 {T_y}^2 - \frac{\triangle x^2 \triangle y^2}{{F_{i,j}}^2} \end{array} \tag{3.12}$$

Se puede generalizar para N dimensiones lo hecho para 2D. Si se tomara un espaciado entre celdas igual para cualquiera de las dimensiones $\triangle x = \triangle y = \triangle z = ... = h$ y denotando T_d como la generalización de T_x y T_y para cualquier dimensión d, hasta N los coeficientes de la ecuación de segundo grado serían:

$$\begin{array}{c} a = N \\ b = \sum_{d=1}^{N} T_d \\ c = \left(\sum_{d=1}^{N} {T_d}^2\right) - \frac{h^2}{F^2} \end{array} \tag{3.13}$$

Para el caso en el que $N = 2$:

$$\begin{aligned} a &= 2 \\ b &= -2(T_x + T_y) \\ c &= ({T_x}^2 + {T_y}^2) - \frac{h^2}{F^2} \end{aligned} \tag{3.14}$$

Sabiendo que la solución a la ecuación de segundo grado es:

$$T = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{3.15}$$

Sustituyendo los valores de a, b y c en (3.15) se llega a:

$$T = \frac{T_x + T_y}{2} \pm \frac{\sqrt{\frac{2h^2}{F^2} - (T_x - T_y)^2}}{2} \tag{3.16}$$

La onda que se propaga cumple el principio de causalidad. Es decir, para llegar a un punto de la malla con mayor tiempo de llegada, se ha de pasar por los puntos vecinos con un menor tiempo de llegada. De no cumplirse esto, implicaría que en algún punto el frente de velocidades F sería negativo ($F \geq 0$ para cualquier punto es una condición del método FM).

La solución presentada en (3.16) solamente respeta la condición de causalidad si:

$$T \geq |T_x - T_y| \tag{3.17}$$

No es difícil ver que dicha condición se cumple únicamente para (3.16) cuando:

$$|T_x - T_y| \leq \frac{h}{F} \tag{3.18}$$

En el caso de que la condición (3.18) no se cumpla, el valor de T vendrá determinado por una relación que no tiene en cuenta T_x y T_y, sino solamente uno de los dos. Es la conocida como *one sided update*:

$$T = min\,(T_x, T_y) + \frac{h}{F} \tag{3.19}$$

3.3. Algoritmo de Dijkstra

3.3.1. Introducción teórica

Hasta ahora se ha visto un enfoque intuitivo del algoritmo de Fast Marching, así como la solución discreta de la ecuación Eikonal, parte importante del mencionado algoritmo. Pero si se quiere entender con algo más de detalle como funciona el algoritmo FM se debe empezar por entender, aunque sea de manera general, el algoritmo de Dijkstra (explicaciones y códigos basados en [19]) .

Se parte de la idea de que se desea obtener el mapa de distancias D, que recoge las distancias a cada nodo de una malla discreta al punto o puntos iniciales. Para calcular dichos valores se tiene otro mapa de "costes", es decir un mapa W que contiene el coste métrico o distancia que se debe recorrer para llegar de un punto a otro. De esta manera, si se quisiera conocer el valor de la distancia en un punto j a partir de la distancia conocida de un nodo vecino k se haría del siguiente modo:

$$D_j \leftarrow \min_{k \sim j} D_k + W_j. \tag{3.20}$$

Para calcular los valores de D de una manera ordenada y correcta el algoritmo clasifica los diferentes nodos en 3 tipos diferentes, siendo entre ellos mutuamente excluyentes (un nodo no puede ser de dos tipos a la vez):

- **Congelados**: Puntos cuyo valor de D ya ha sido calculado y dicho valor ya no va a cambiar. Puntos que han sido ya alcanzados por el frente de onda.
- **Banda estrecha**: Puntos que pasarán a formar parte del frente de onda en la próxima iteración. Tienen un valor asignado para D calculada mediante la expresión (3.20), pero este valor podría aún cambiar en futuras iteraciones del algoritmo.
- **Desconocidos**: Puntos que el frente de onda todavía no ha alcanzado. Tiene lo equivalente a un valor de ∞ hasta que sean evaluados y pasen a formar parte de la banda estrecha.

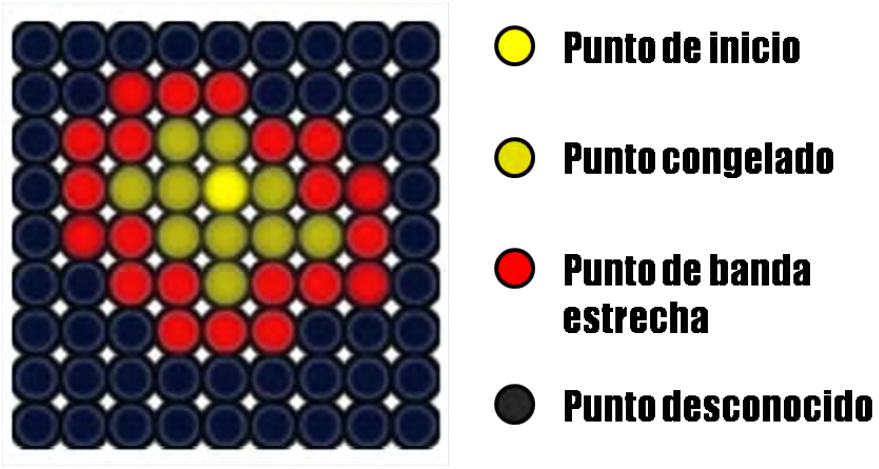


Figura 3.2: Representación gráfica de los diferentes tipos de puntos

3.3.2. Ejemplo práctico

Para una mejor comprensión de como funciona el algoritmo se va a presentar un ejemplo muy sencillo del algoritmo programado en Matlab (C.1).

Esencialmente el algoritmo consta de 3 matrices:

- **Matriz** W: Matriz que contiene los costes de viajar de un nodo a otro. Para sencillez del ejemplo en este caso, todos los valores valen 1. Es un *input* del algoritmo, y su valor no cambia con cada iteración.
- **Matriz** S: Matriz que contiene la información sobre en qué estado está cada punto, si se trata de un punto congelados o muertos (-1), perteneciente a la banda estrecha (1) o aquellos desconocidos (0). Sus valores se actualizan en cada iteración.
- **Matriz** D: Matriz que contiene las distancias que se quieren calcular. Inicialmente los puntos desconocidos están a una distancia infinita y en cada iteración los valores se actualizan.

En la tabla 3.1 se presenta el resultado para un ejemplo sencillo, un caso bidimiensional con matrices de 3x3.

Número de iteración	**0**	**1**	**2**	**3**
Matriz S	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ -1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 1 & 1 \\ 1 & 0 & 0 \\ -1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{pmatrix}$
Matriz D	$\begin{pmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ 0 & \infty & \infty \end{pmatrix}$	$\begin{pmatrix} 1 & \infty & \infty \\ 1 & \infty & \infty \\ 0 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 2 \\ 1 & \infty & \infty \\ 0 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 2 \\ 1 & 2 & 2 \\ 0 & 1 & 1 \end{pmatrix}$

Tabla 3.1: Evolución de los valores de las matrices S y D con cada iteración

3.4. Algoritmo de Fast Marching

3.4.1. Introdución teórica

Observando los resultados de la tabla 3.1 no es difícil llegar a la conclusión de que la manera de calcular las distancias mediante la expresión (3.20) no es óptima. De ahí que la principal diferencia entre el algoritmo de Fast Marching y el algoritmo de Dijkstra sea que, en FM para el cálculo en cada nodo del mapa de distancias se utilice la solución discreta a la ecuación Eikonal presentada en la sección 3.2.3 .

Lo primero que se debe hacer para encontrar una expresión que sustituya a (3.20) es encontrar un paralelismo entre las matrices del algoritmo de Dijkstra W, y D y las variables usadas en la sección 3.2 para la resolución de la ecuación Eikonal. Es fácil ver que la matriz D corresponderá a la variable de tiempo (mayor cuanto más lejos del inicio estén los puntos del mapa) y la matriz de costes W equivaldrá al cociente del espaciado entre celdas entre la función de velocidades $\frac{h}{F}$. De esta manera FM sustituirá (3.20) por:

$$D_k = \begin{cases} \frac{d_x + d_y + \sqrt{2{W_k}^2 - (d_x - d_y)^2}}{2} & \text{si} \quad |d_x - d_y| \leq W_k \\ min(d_x, d_y) + W_k & \text{Para cualquier otro caso} \end{cases} \tag{3.21}$$

Donde:

$$d_x = \text{mín}(D_{k+1,\ell}, D_{k-1,\ell}) \quad \text{y} \quad d_y = \text{mín}(D_{k,\ell+1}, D_{k,\ell-1}). \tag{3.22}$$

3.4.2. Ejemplo práctico

De manera análoga a (3.3) se ha adjuntado en C.2 un código sencillo del algoritmo FM en Matlab. El código es idéntico al del ejemplo de la tabla 3.1 salvando la sustitución de la expresión (3.20) por la ecuación 3.21 a la hora de computar el siguiente nodo de la matriz D. En la tabla 3.2 se representan los resultados de este código para una sencilla matriz de 3x3.

Número de iteración	**0**	**1**	**2**	**3**
Matriz S	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ -1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 1 & 1 \\ 1 & 0 & 0 \\ -1 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{pmatrix}$
Matriz D	$\begin{pmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ 0 & \infty & \infty \end{pmatrix}$	$\begin{pmatrix} 1 & \infty & \infty \\ 1 & \infty & \infty \\ 0 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1{,}7071 & 1{,}7071 \\ 1 & \infty & \infty \\ 0 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1{,}7071 & 1{,}7071 \\ 1 & 1{,}7071 & 1{,}7071 \\ 0 & 1 & 1 \end{pmatrix}$

Tabla 3.2: Evolución de los valores de las matrices S y D con cada iteración

3.5. Inclusión de un campo vectorial

La explicación que se ha dado sobre el algoritmo FM no tiene en cuenta la posible influencia de un campo vectorial. Dicho de otra manera, el método FM original considera

un mapa de costes con potencial 0 [20].

Para representar correctamente el fenómeno de propagación del fuego es evidente que será necesario un método que tenga en cuenta el efecto de variables de carácter vectorial como lo son el viento o la pendiente del terreno.

Explicado de manera genérica, si se denota el cálculo de la matriz D a partir de la matriz de costes en el algoritmo FM como el cálculo de un campo de potencial a partir de variables escalares y adicionalmente se tiene en cuenta otro campo de carácter vectorial, el potencial final vendrá dado por:

$$F_{ij} = F_{scal,ij} + F_{vect,ij} \tag{3.23}$$

Donde $F_{scal,ij}$ representa la influencia del mapa de costes escalar y $F_{vect,ij}$ la de campos vectoriales externos. $F_{vect,ij}$ estará a su vez constituido por la suma de campos vectoriales externos que afecten al proceso (en el caso del incendio, el viento y la pendiente del terreno).

En la figura 3.5 se puede apreciar el efecto de un campo vectorial externo en una propagación calculada por FM.

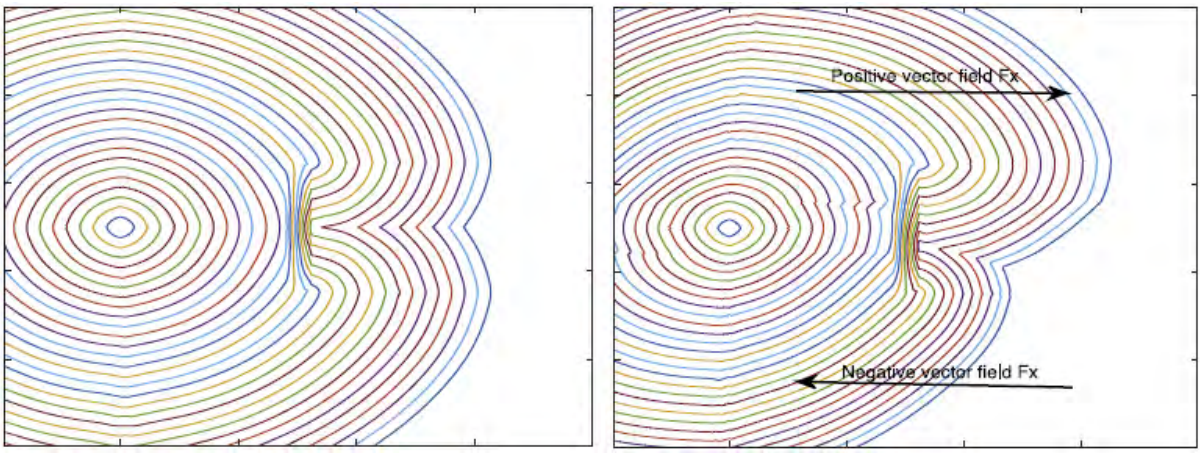


Figura 3.3: Efecto de un campo vectorial externo en el algoritmo FM

Capítulo 4

Desarrollo del Trabajo

La aplicación de incendios forestales desarrollada en este TFG consiste en una interfaz de usuario de Matlab que permite el uso de mapas reales de elevación y de combustible para la simulación de un incendio pudiendo variar parámetros como el tiempo transcurrido o la intensidad del viento. Se busca la mayor sencillez posible para que un usuario sin amplios conocimientos ingenieriles pueda usarlo. Otro de los objetivos principales es la clara visualización del frente de propagación del fuego.

La aplicación se ha desarrollado en el programa Matlab [21], una potente herramienta de computación con aplicaciones matemáticas, de ingeniería, etc.

En la sección 4.1 se pretende dar una introducción a nivel usuario de la aplicación, mostrando los controles de las variables y cómo estos se utilizan. En la sección 4.2 se explica cómo se obtienen mapas de elevación y combustible reales. En la sección 4.3 se explica la programación de la interfaz mediante el GUI de Matlab. Dicho GUI hará uso de una función muy importante para el desarrollo de la interfaz y de la simulación de la programación que se explica en la sección 4.3.

4.1. Explicación introductoria de la aplicación

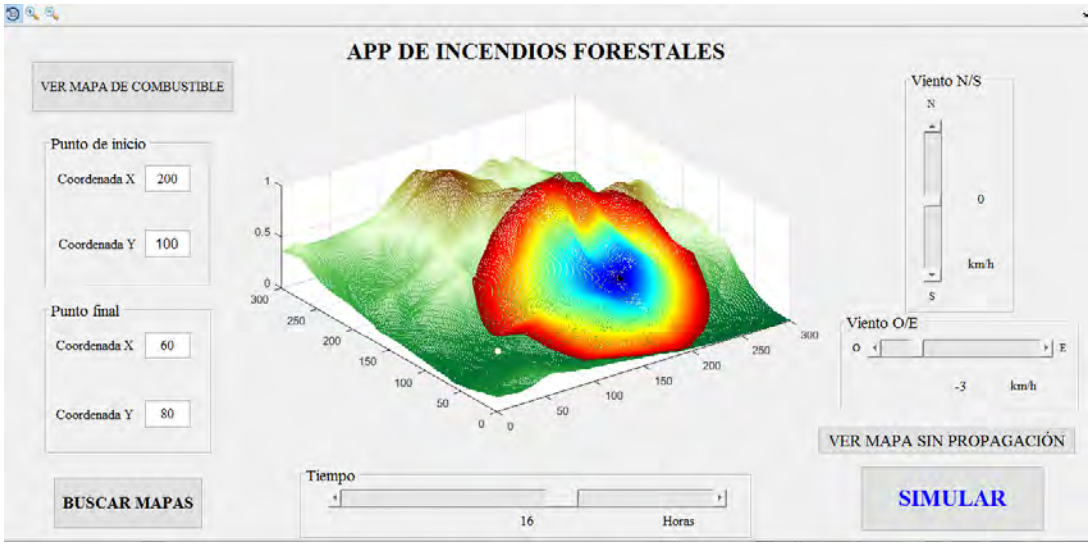


Figura 4.1: Imagen de la interfaz

El usuario puede interactuar con la interfaz mediante la selección de varios parámetros como se puede intuir en la figura 4.1. A continuación se explica el funcionamiento de cada uno de los parámetros y botones:

- **Selección de mapas**: Este botón permite la selección de un archivo .mat que contenga dos matrices con información relativa al terreno sobre el cual se quiere

simular la propagación del fuego. Contendrá una matriz denominada "elevacion" con información relativa a la elevación del terreno y otra matriz denominada "velocidad" que corresponde al mapa de flamabilidad. El mapa de flamabilidad debe tener valores entre 0 y 1, donde 0 implica terreno no flamable y 1 terreno sobre el cual el fuego podrá arder con máxima facilidad. Conviene que el usuario tenga en cuenta que la primera acción a realizar es la selección de mapas, ya que de lo contrario podrá haber un error en la aplicación.

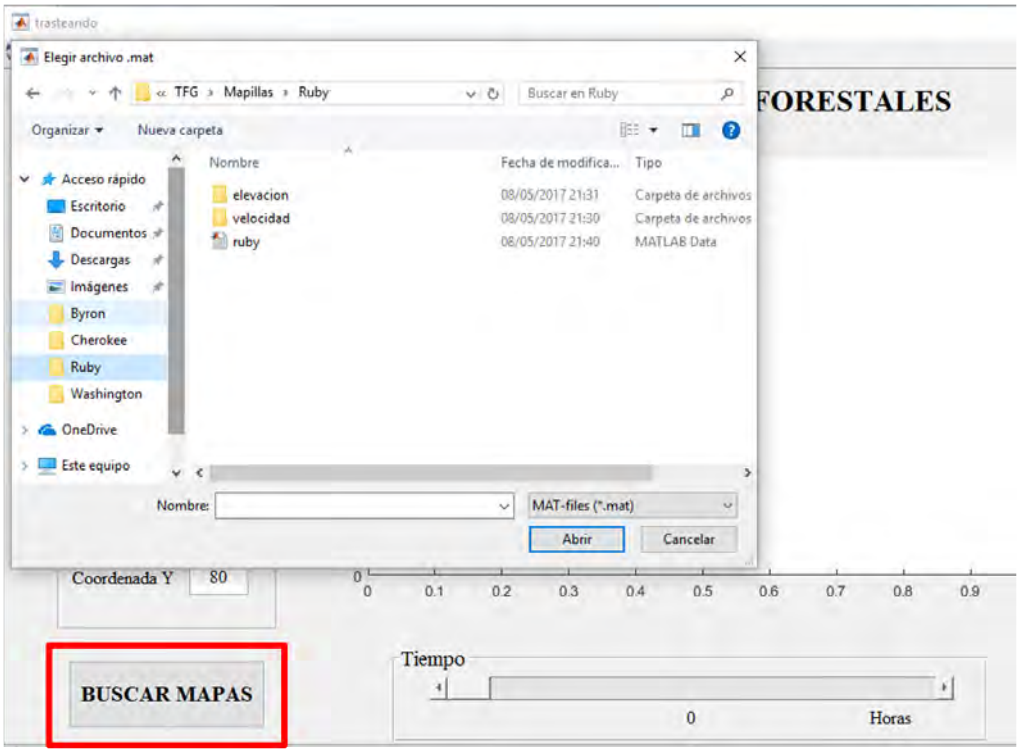


Figura 4.2: Ventana que le aparece al usuario al pulsar el botón "Buscar mapas"

- **Visualización del mapa de elevación sin la propagación del fuego**: Este botón permite al usuario la visualización en 3D del mapa sobre el cual se va a propagar el fuego. De esta manera le permite familiarizarse con las dimensiones y coordenadas del mapa para así poder elegir de manera acertada el resto de parámetros necesarios para la simulación del incendio.

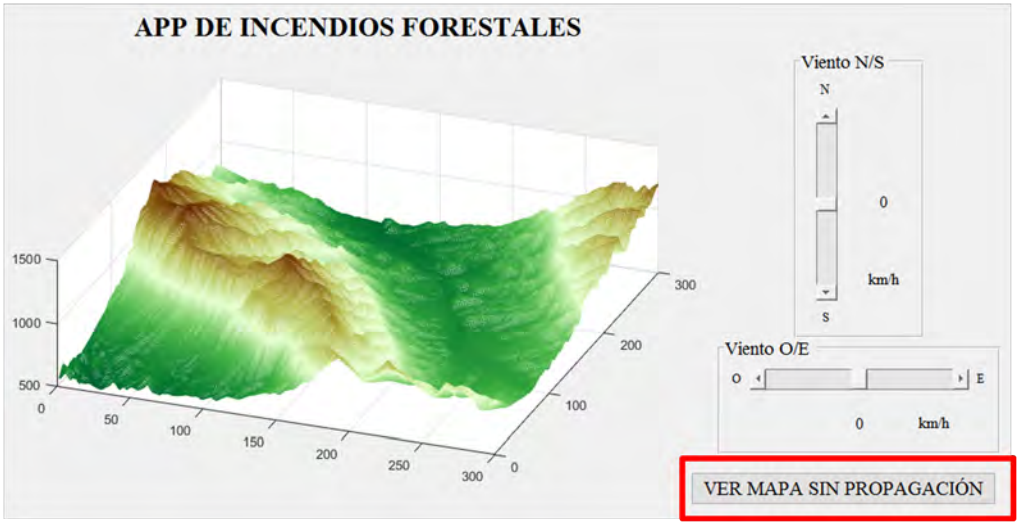


Figura 4.3: Visualización de un mapa sin la propagación del fuego

- **Selección de coordenadas de puntos de inicio y final del fuego**: El usuario debe introducir las coordenadas del punto en el cual se origina el fuego en la simulación y un punto de interés que será el punto final de la simulación (en un caso práctico se puede tratar de una zona de interés: como una zona habitada o la cual es importante que no alcance el fuego). En el gráfico se representan los puntos incial y final con puntos negro y blanco respectivamente.

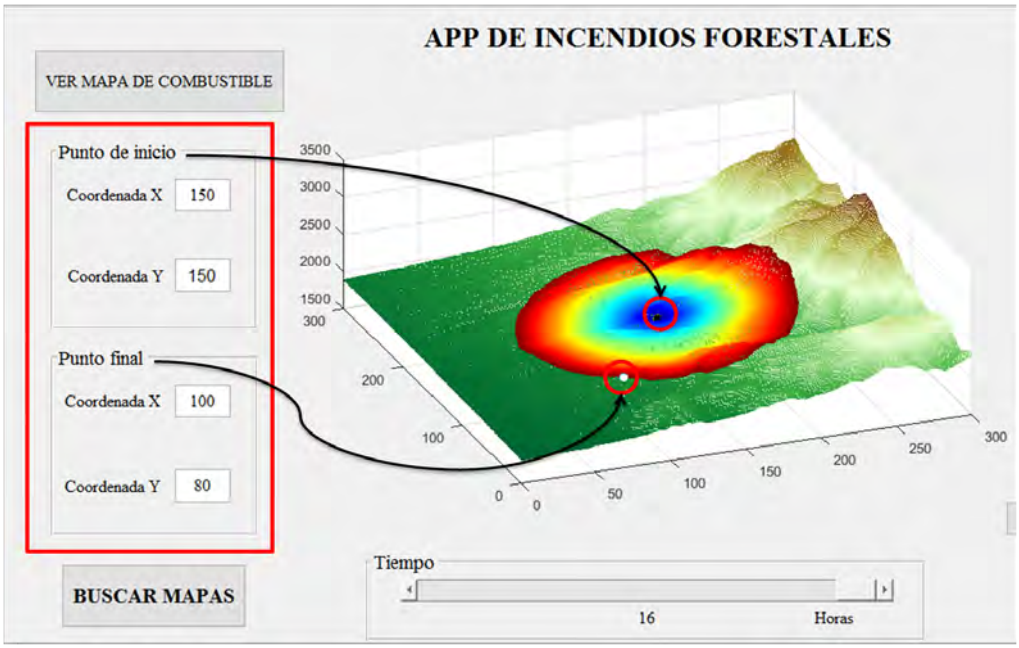


Figura 4.4: Localización de los puntos de selección de coordenadas en la aplicación

- **Selección del valor del tiempo de propagación**: Dadas las coordenadas de inicio y fin que ha seleccionado el usuario, la aplicación permite ver la propagación del

fuego en momentos intermedios desde el momento que empieza hasta el momento que llega al punto final. Esto se controla mediante el *slider* de tiempo de la aplicación. Si el valor máximo del slider corresponde a lo que tarda, en teoría, el fuego en llegar al punto especificado como punto final, entonces mediante la selección de un punto intermedio del slider se verá cómo se comporta la propagación antes de llegar al punto de fin.

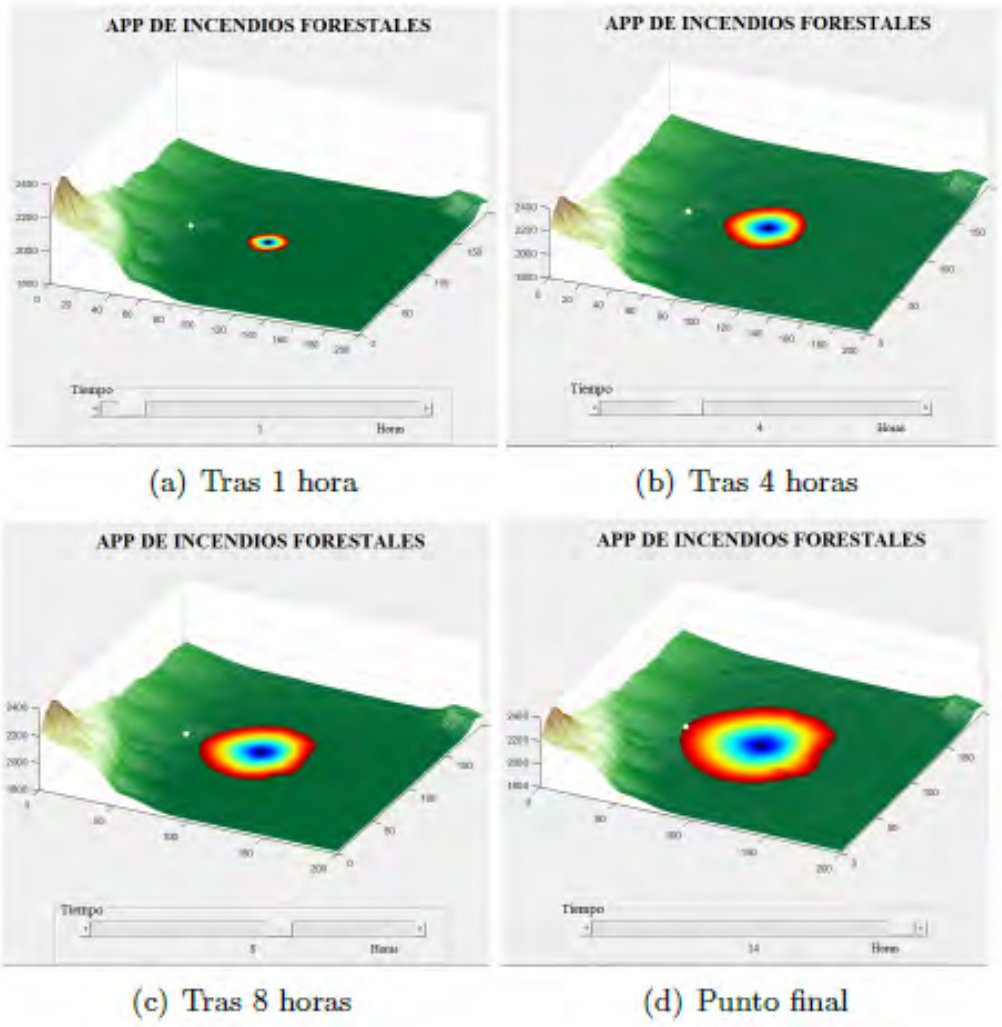


Figura 4.5: Propagación del fuego conforme pasa el tiempo

- **Selección del valor del viento**: Al usuario se le permite también incluir el valor de un viento constante dirección Norte/Sur (correspondiente al eje Y del gráfico) y otro en dirección Este/Oeste (correspondiente al eje X del gráfico). La existencia de un viento en cada una de estas direcciones, o en una combinación de ambas (viento suroeste, por ejemplo) afectará, como es evidente, a la simulación de la propagación del fuego.

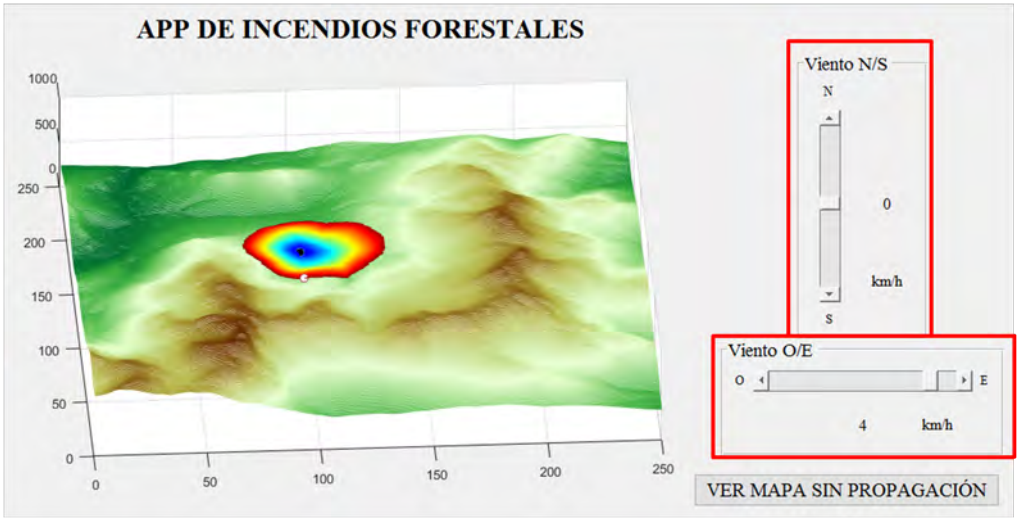


Figura 4.6: Cursores donde variar los valores del tiempo

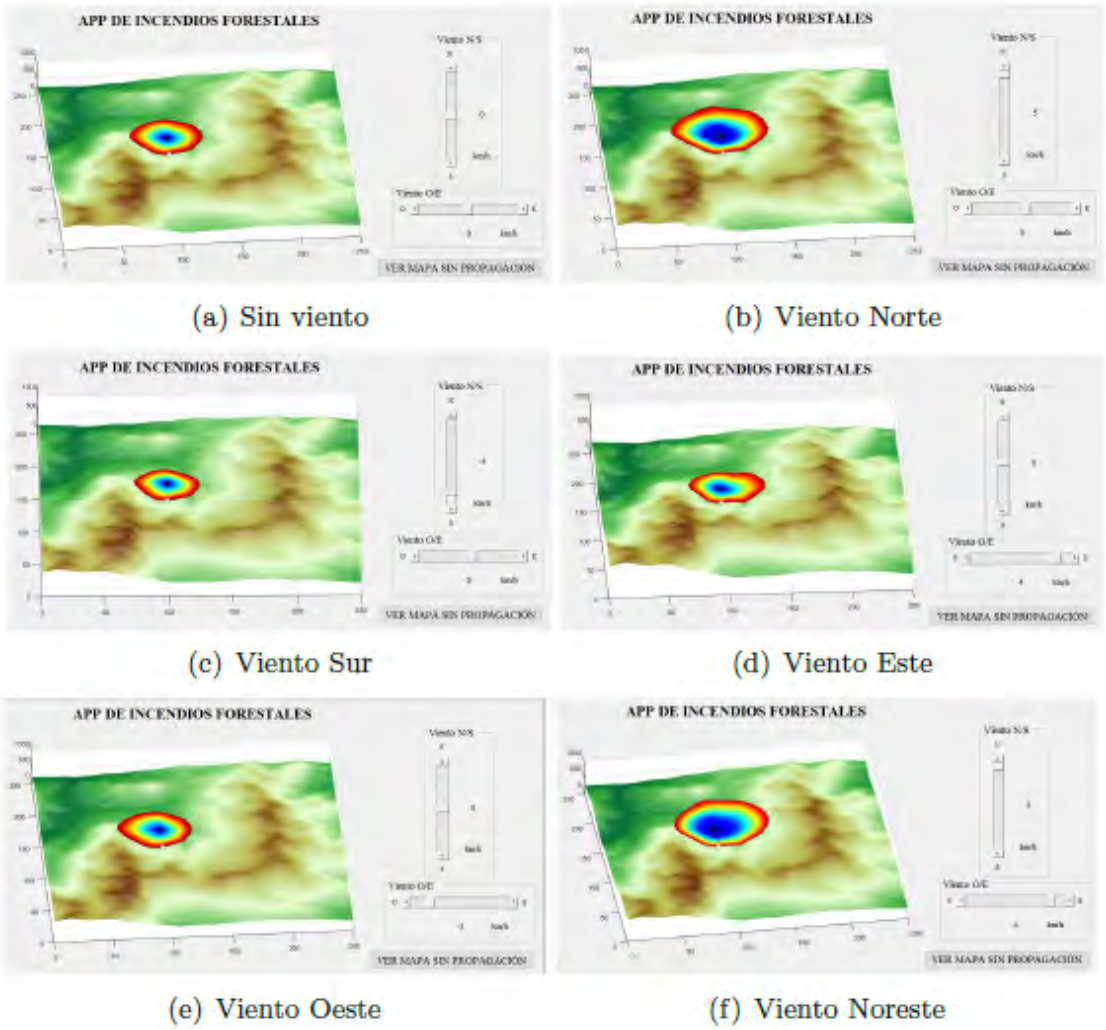
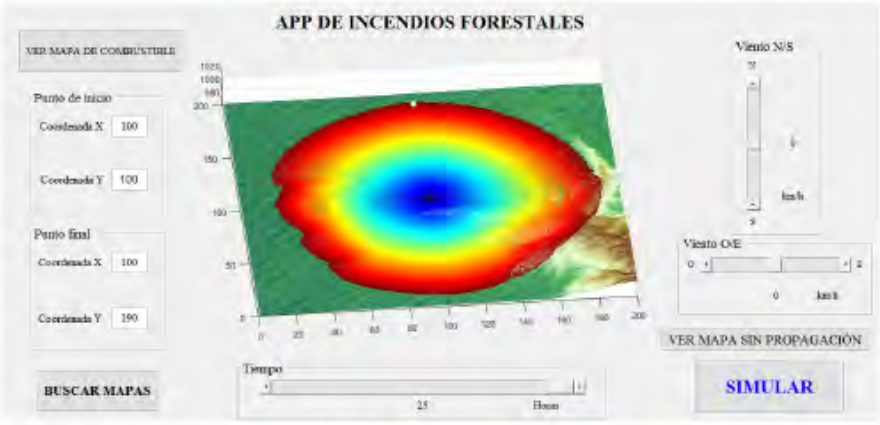


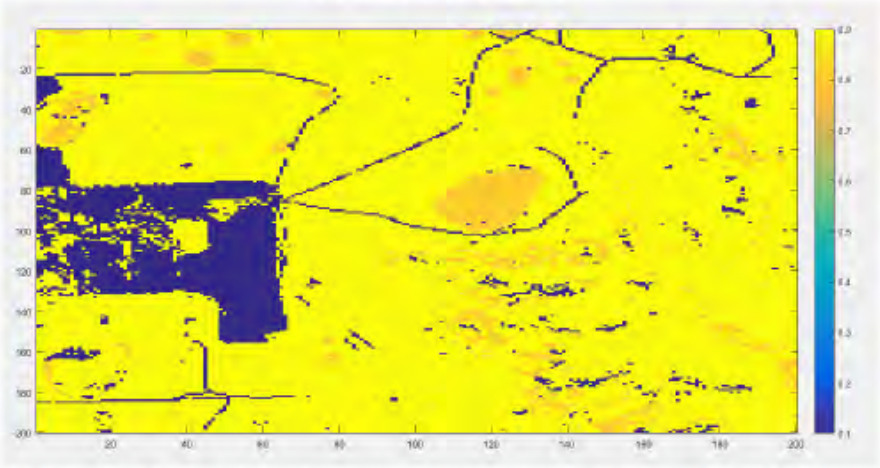
Figura 4.7: Variación en la propagación del fuego en función de la variación del viento

- **Visualización del mapa de combustible**: La aplicación también permite al usuario visualizar el mapa de combustible o flamabilidad con la misma orientación que el mapa de elevación, para, de esta manera, poder entender mejor la propagación del fuego, y elegir de manera más informada el resto de parámetros, como el punto de inicio o final del fuego.

En la figura 4.8 se puede apreciar como el fuego se propaga con mayor rapidez hacia la zona montañosa (derecha del mapa) que hacia la zona más plana. Si se observa el mapa de combustible puede deducirse que esto es una consecuencia lógica del hecho de que en la zona de la izquierda del mapa hay una zona con mínima flamabilidad (probablemente un lago o zona húmeda).



(a) Mapa de propagación



(b) Ventana del mapa de combustible que aparece cuando se pulsa el botón "Ver mapa de combustible"

Figura 4.8: Conjunto de mapa de propagación y mapa de flamabilidad

- **Botón de simular**: Cada vez que se cambia alguno de los parámetros anteriores: viento, tiempo, coordenadas, etc. será necesario pulsar este botón para actualizar los resultados gráficos.

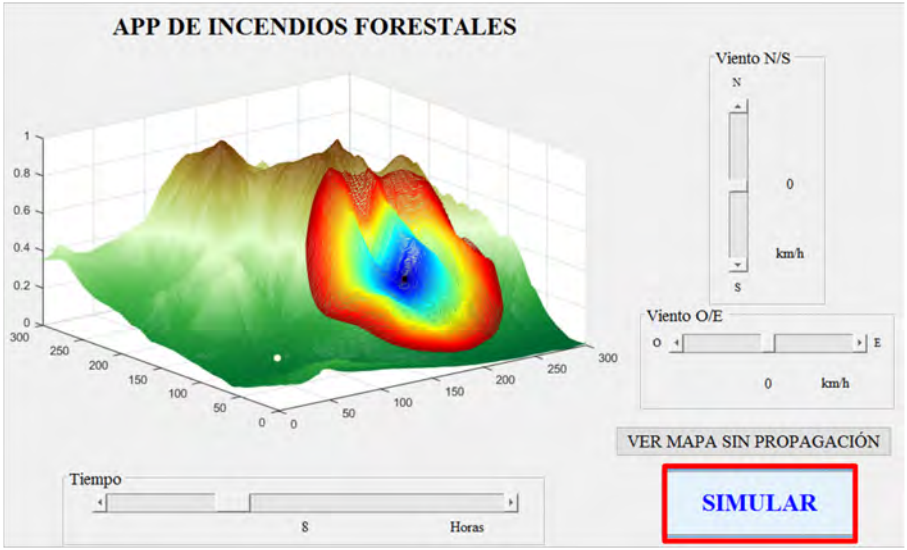


Figura 4.9: Localización del botón de inicio de simulación de la propagación

4.2. Mapas de elevación y velocidad

Para el correcto funcionamiento de la aplicación se deberá seleccionar un fichero .mat que contenga dos matrices en Matlab, elevacion y velocidad, con información correspondientes a la elevación y a la flamabilidad del terreno. A continuación se explicarán más en profundidad estos dos tipos de matrices:

- **Mapa de elevación**: Se trata de una matriz cuadrada, nxn, pudiendo ser n en teoría cualquier número entero, aunque en la práctica, para que el algoritmo FM funcione correctamente se recomienda que no sea de más de 400x400. El valor en cada celda de la matriz corresponde a la altura en el eje z que tenga ese punto. El comando `mesh` de matlab permite la representación de una matriz de este tipo.

- **Mapa de velocidad**: Este será la matriz que se introducirá como matriz de costes en el algoritmo FM. Se trata de una matriz con las mismas dimensiones que la matriz elevación, pero en este caso el valor de cada celda de la matriz corresponde a flamabilidad de dicho punto.

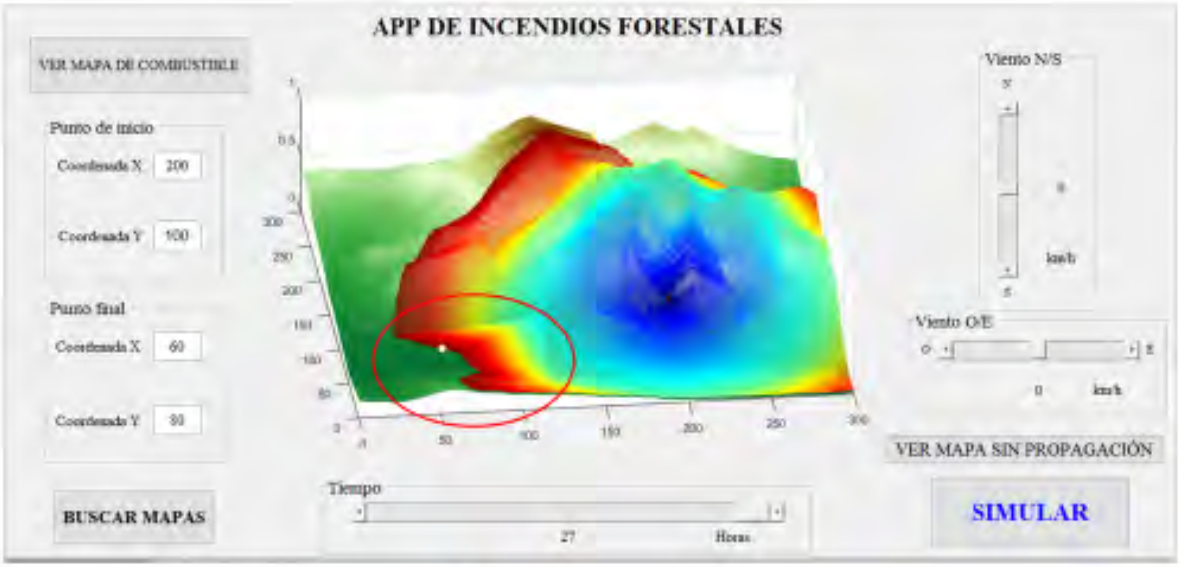
  Los valores de la matriz deberán estar entre 0 y 1 para que el algoritmo funcione correctamente. Un punto con valor 0 es un punto por el que el fuego no puede pasar y aquel con valor 1 es aquel por el cual el fuego pasará con mayor velocidad. Pese a que, en teoría el algoritmo debería poder tratar con los valores extremos, 0 y 1, a la hora de la verdad, teniendo en cuenta que el algoritmo nunca permite que el frente retroceda es peligroso trabajar con mapas con varios puntos con valor 0, ya que es posible que el algoritmo FM falle y ocurra un error en la aplicación. De manera análoga puede suponer un problema para el algoritmo tener mapas de

velocidad con valores muy diversos esparcidos de manera aleatoria ya que es posible que el frente tienda a intentar retroceder.

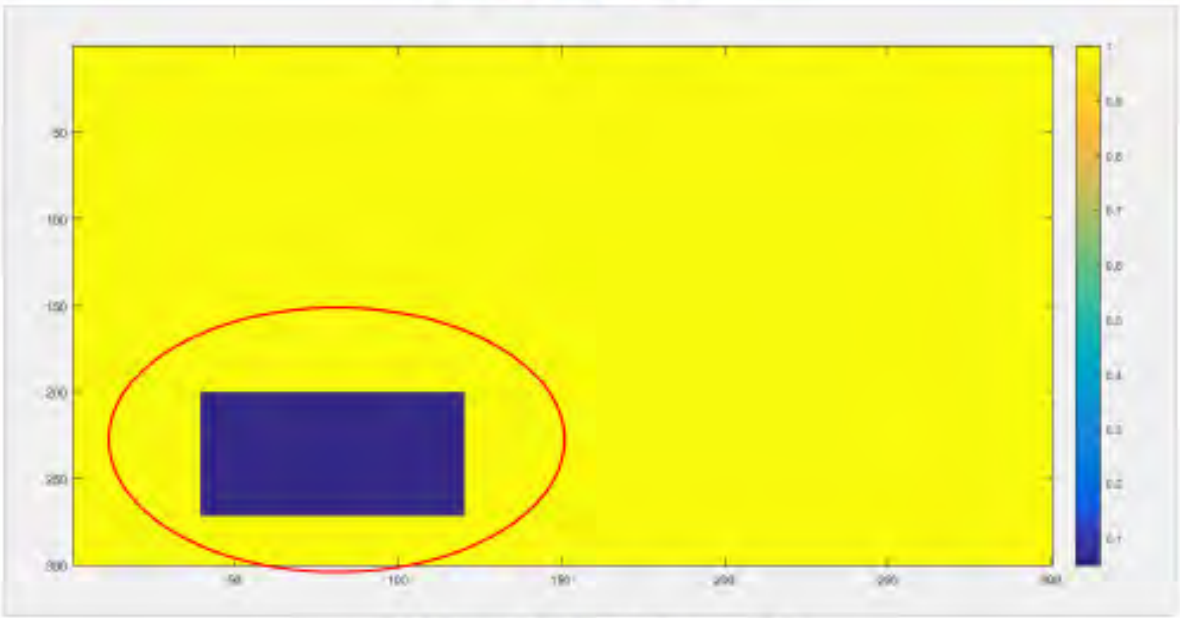
Para evitar problemas causados por el mapa de velocidad en la aplicación, la matriz que contenga información de flamabilidad real deberá ser tratada para intentar minimizar los problemas que se puedan originar al ejecutar el algoritmo FM. Este tratamiento consiste, esencialmente en reescalar los valores de la matriz para evitar valores muy cercanos a 0 y en eliminar la variabilidad de valores mediante redondeos a la primera cifra decimal.

4.2.1. Prueba con mapa de velocidad ficticio

Cualquier mapa de velocidad podría usarse en la aplicación asociado a uno de elevación. Al inicio de este proyecto, para verificar el correcto funcionamiento del algoritmo y de la aplicación se hicieron pruebas usando la conocida matriz de Washington (que representa el monte Washington, USA) como matriz de elevación y un mapa de velocidades constante, con excepción de un rectángulo en el cual los valores son muy bajos. El resultado se puede apreciar en la figura 4.10.



(a) Mapa de propagación



(b) Mapa de velocidad inventado

Figura 4.10: Monte Washington con mapa de velocidades inventado

4.2.2. Mapas de elevación y velocidad reales

El hecho de que la aplicación se comporte de la manera esperada para un mapa de velocidades ficticio es una buena señal. Pero, evidentemente, para que la aplicación tenga un carácter práctico a nivel de ingeniería se necesita que pueda ser usada para mapas de elevación y velocidad reales. Es decir que logre simular sobre un terreno real la propagación del fuego basado en un mapa de flamabilidad real.

Desgraciadamente España carece de bases de datos con mapas de elevación y de combustible fácilmente accesibles. En [22] se pueden encontrar mapas de elevación, vegetación o flamabilidad de prácticamente cualquier punto de los Estados Unidos.

El proceso a seguir para descargar datos es el siguiente:

1. Entrar la parte de la página "Data Distribution Site" y elegir el estado del cual se quieren obtener los mapas.

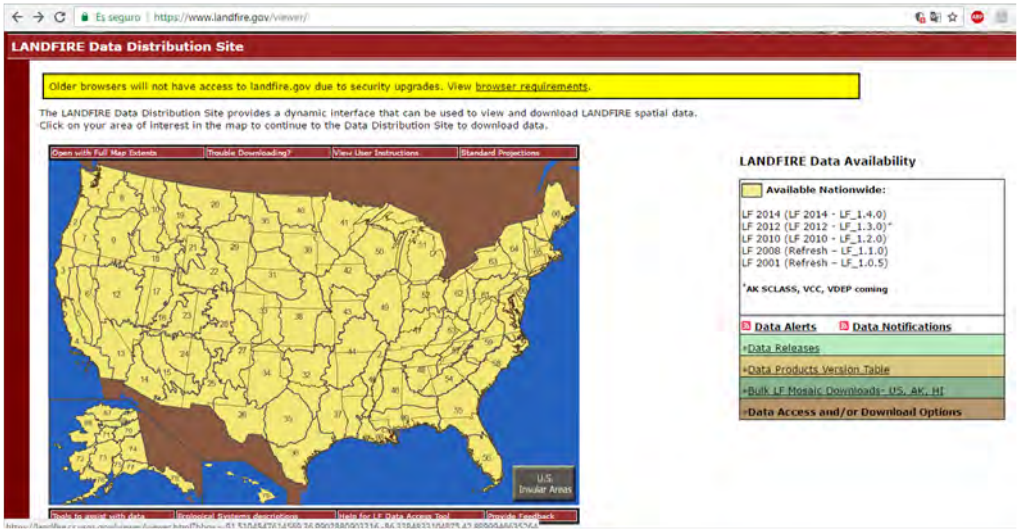


Figura 4.11: Data Distribution Site

2. Seleccionar la opción "Download Data", seleccionar los mapas que se quieran descargar (en nuestro caso "Fuel model maps" y "elevation") y seleccionar la porción del mapa que se quiera guardar.

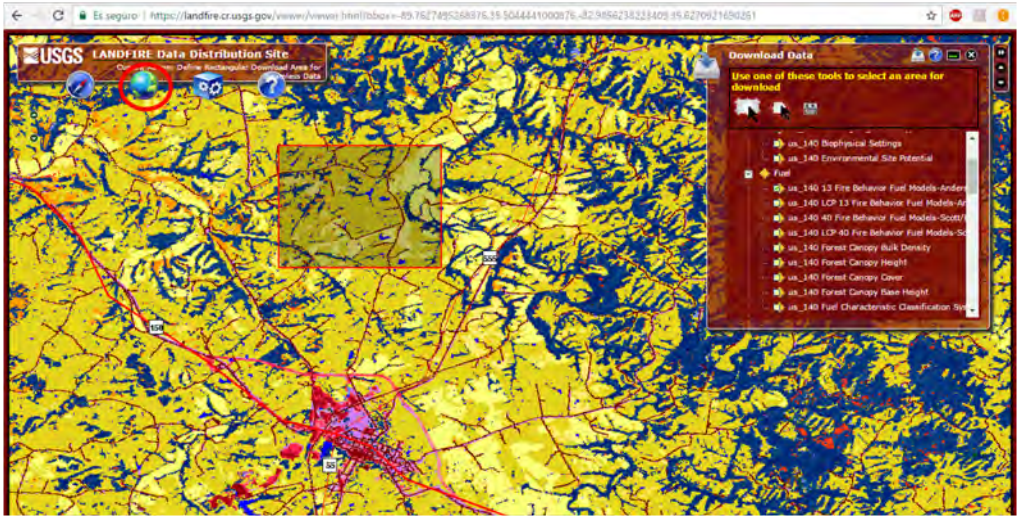


Figura 4.12: Mapa donde seleccionamos la información que necesitemos

3. En la opción "Modify Data Request", seleccionar que el tipo de datos a descargar sean de tipo GeoTiff, ya que este formato es mucho más fácil de llevar al entorno de Matlab.

Figura 4.13: Entorno donde modificar el formato de datos a descargar

4. Descargar los datos.

Una vez se tienen los datos reales descargados el proceso para convertirlos en las matrices elevacion y velocidad en Matlab es bastante sencillo. Esencialmente consiste en:

1. Convertir los mapas en formato .tiff a matrices de matlab y redimensionarlas según se considere conveniente.

2. El mapa de combustible que se descarga contiene valores de 0 a 99, siendo 0 muy inflamable y 99 nada flamable. Evidentemente se necesitará un reescalado de valores de 0 a 1 de menos a más flamable para poder ser usado en el algoritmo.

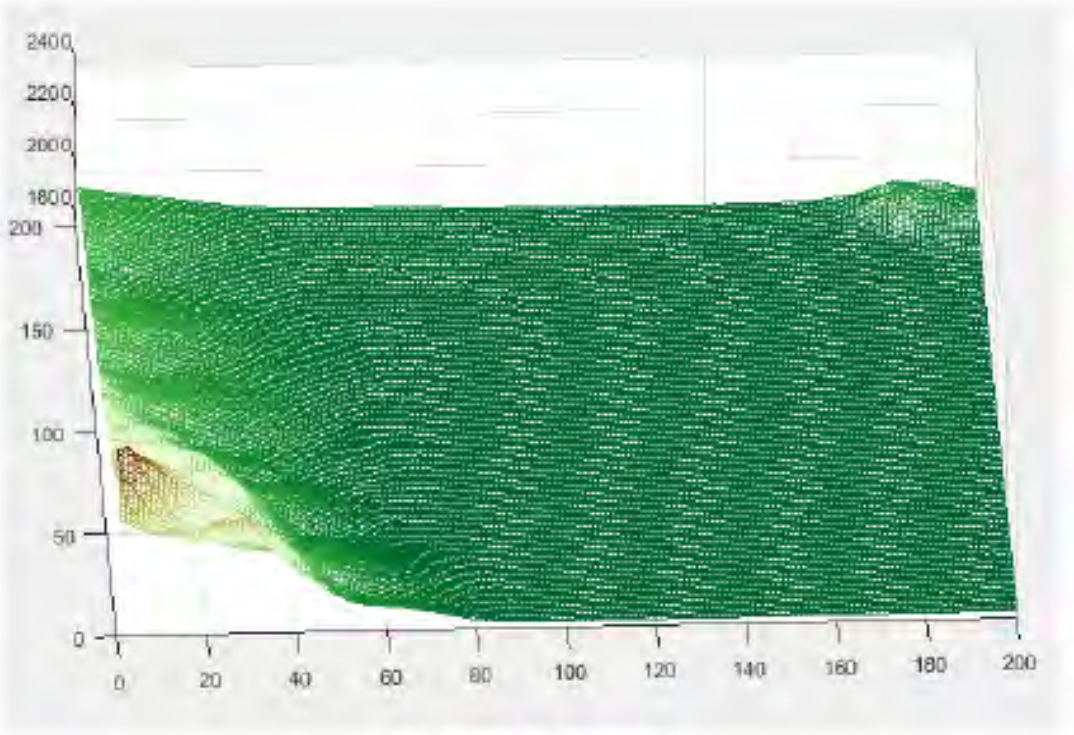
3. Se tendrán que reescalar los datos nuevamente para evitar que haya celdas que contengan valores muy próximos a 0 para la matriz de velocidad, así como redondear los datos al decimal más cercano para evitar excesiva heterogeneidad en los valores de la matriz de velocidad.

A continuación se encuentra un modelo de script de Matlab debidamente comentado para obtener el *workspace*, entorno de variables de Matlab, con las matrices elevacion y velocidad a partir un mapa descargado de la página.

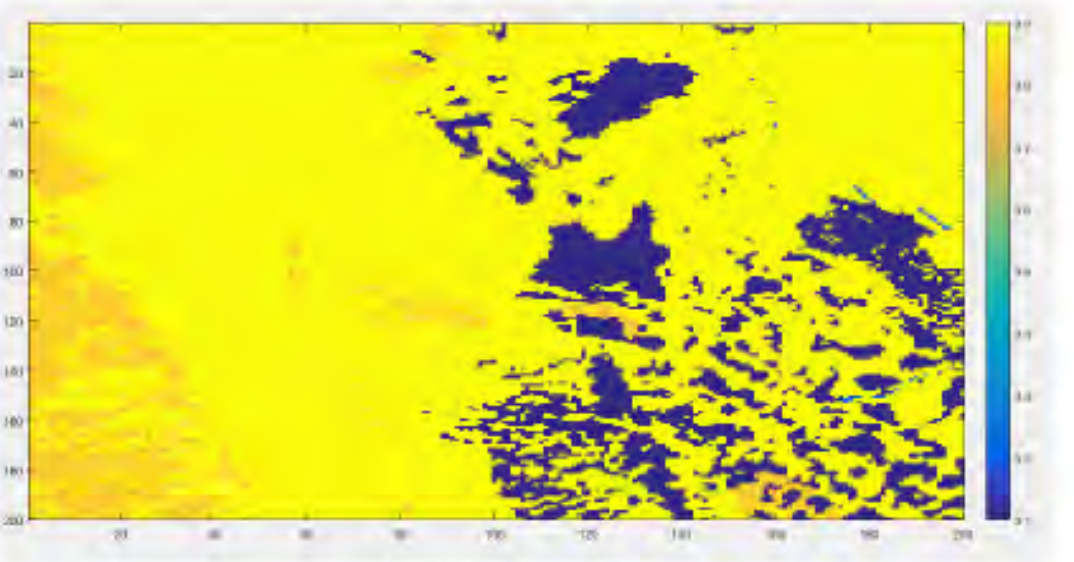
```
1  %Script para la lectura del mapa Ruby
2
3  clear all
4
5  a = double(imread('ruby.tif')); %Lectura del mapa de elvación
6
7  %imread devuelve la matriz con información sobre la imagen leída.
8  % Los números son de tipo int, y para poder reescalarlos nos ...
       interesa convertirlos a tipo double
9
10 elevacion=a(1:200,1:200); %Seleccionamos una sección del mapa completo
11
12 b= double(imread('ru.tif')); %Lectura del mapa de velocidad
13
14 velocidad=b(1:200,1:200); %Sección
15
16 %Reescalado y redondeado de la matriz velocidad
17
18 velocidad=ones(200)-rescale(velocidad,0.05,0.95);
19
20 velocidad=round(rescale(velocidad,0.1,0.9),1);
```

4.2.3. Muestra de mapas reales

En las figruras 4.16, 4.14, 4.17, 4.18 y 4.15 se muestran los mapas de elvación en 3D y de combustible en 2D, obtenidos de la página de Landfire, que se han utilizado para hacer pruebas con la aplicación. Se trata de mapas de entornos forestales reales, cada uno de ellos lleva indicadas las coordenadas de situación.

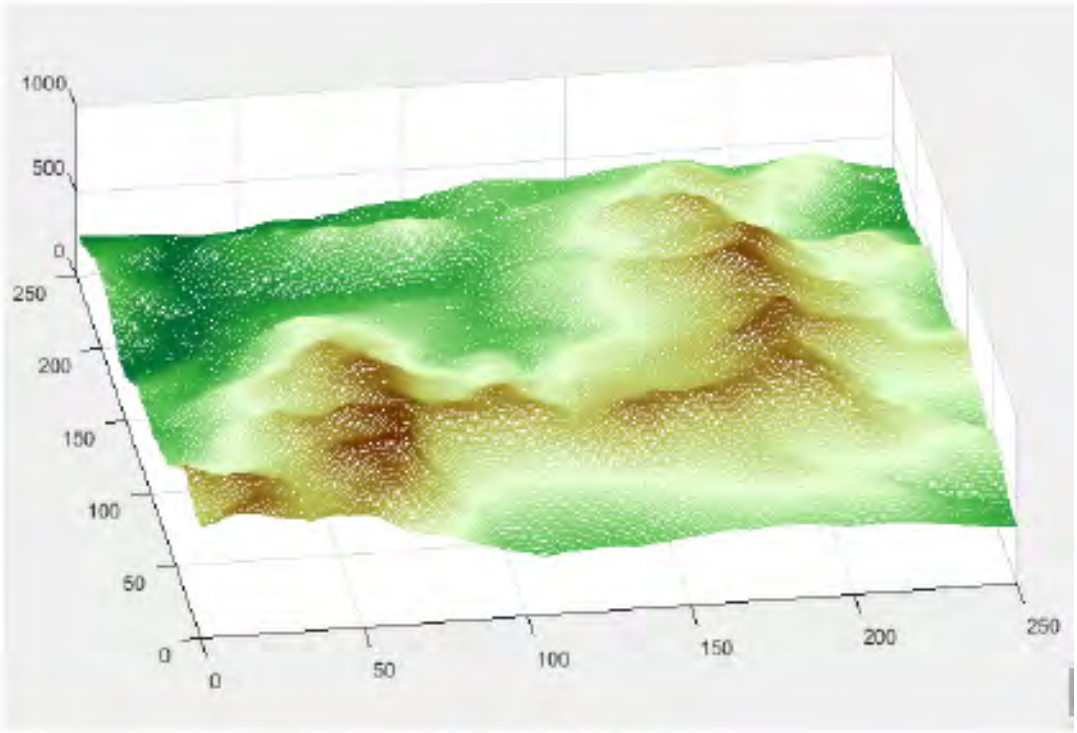


(a) Mapa de elevación

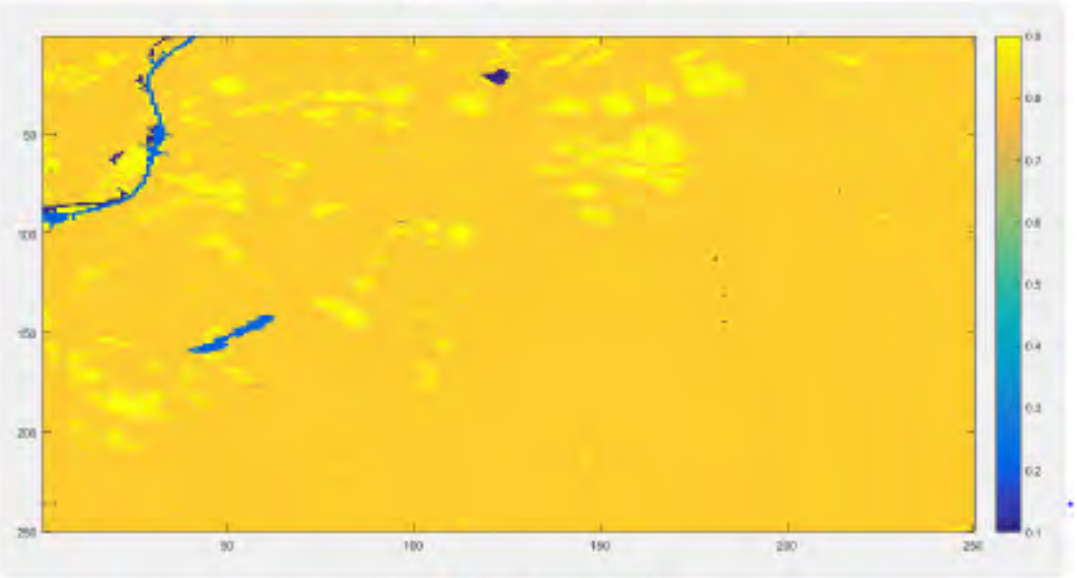


(b) Mapa de velocidad

Figura 4.14: Coordenadas: 40.06, -115.5

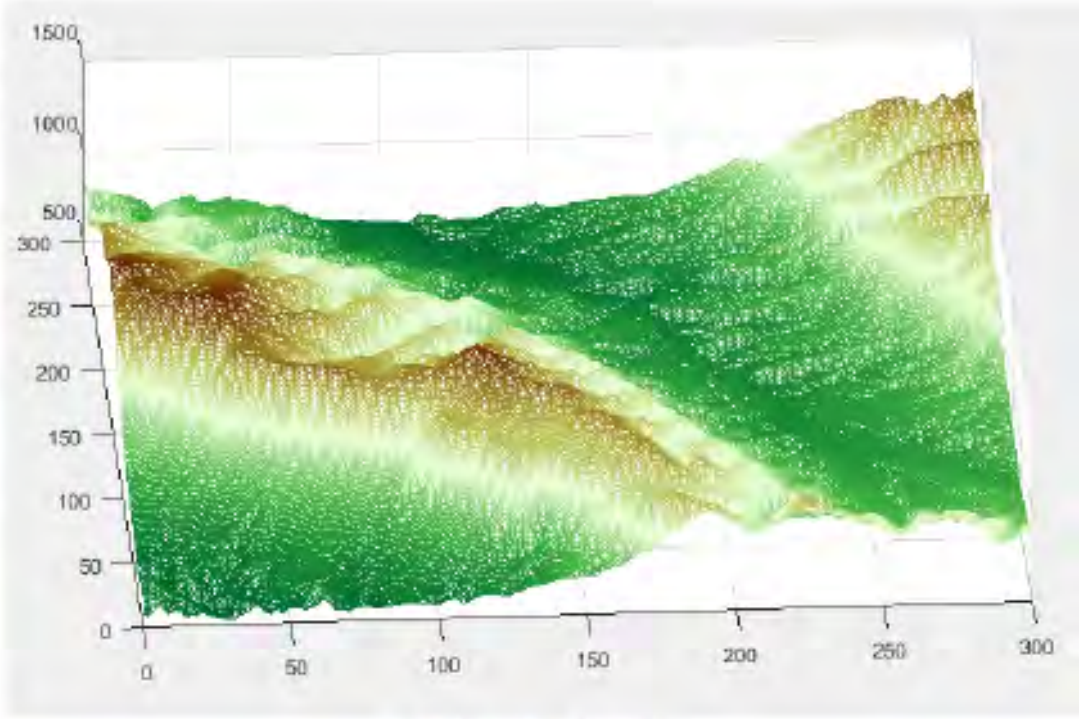


(a) Mapa de elevación

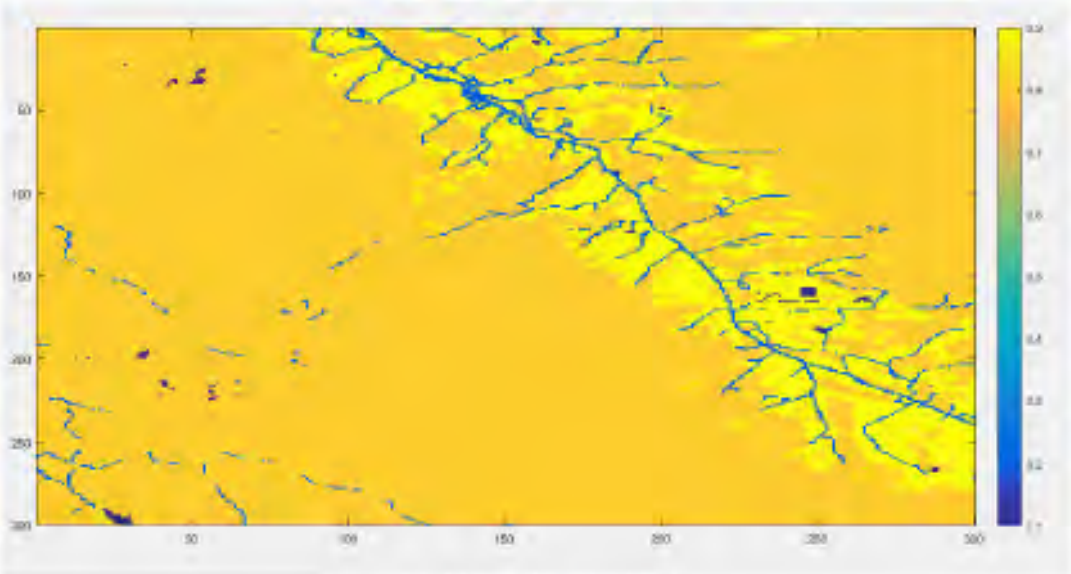


(b) Mapa de velocidad

Figura 4.15: Coordenadas: 44.66, -70.55

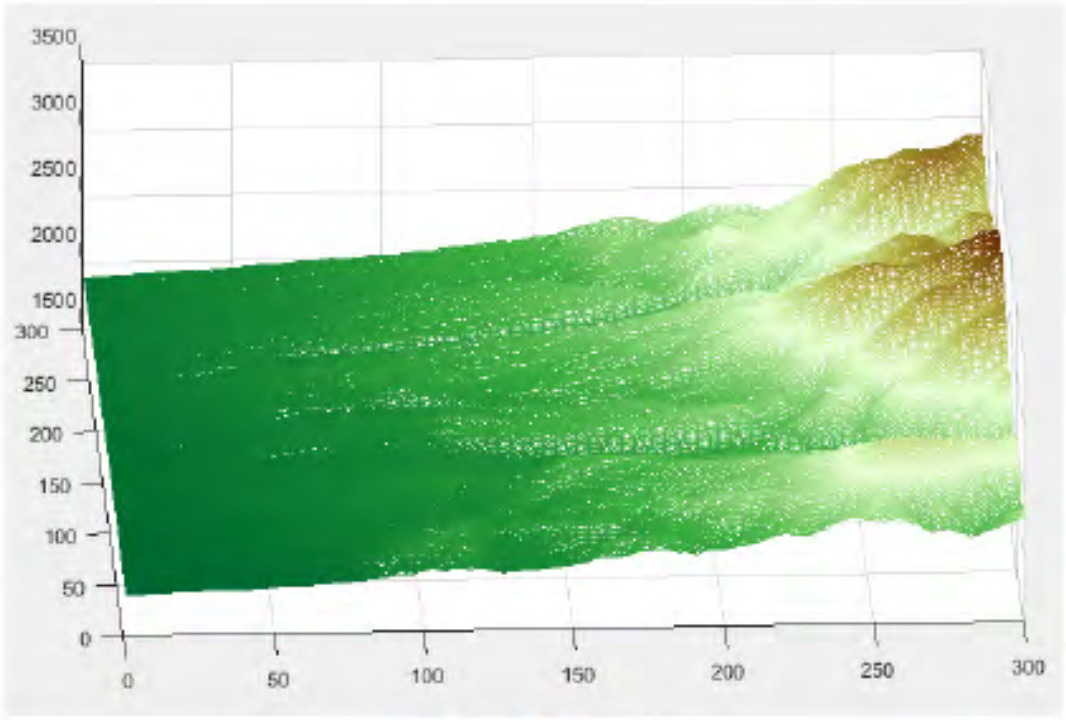


(a) Mapa de elevación

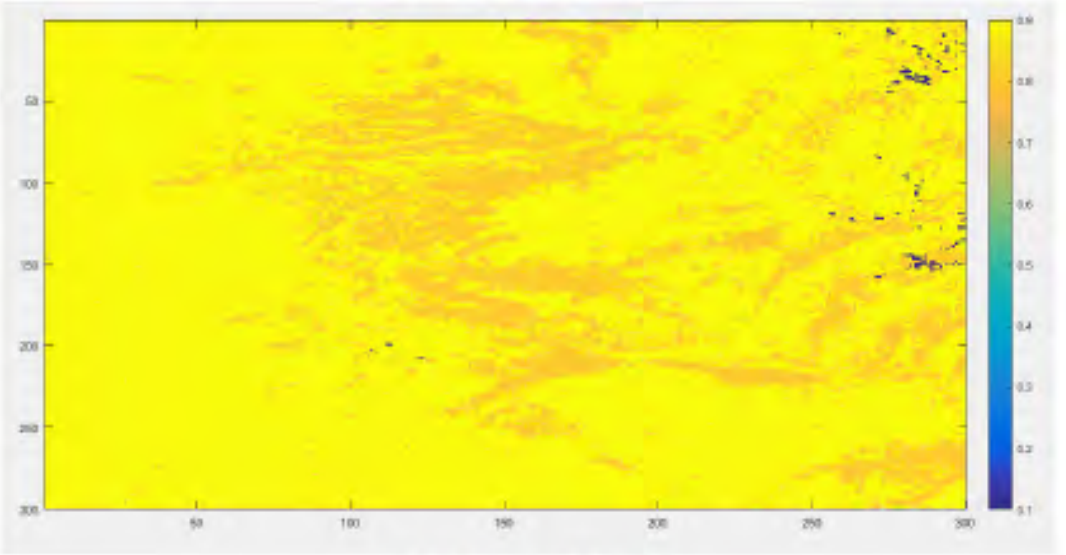


(b) Mapa de velocidad

Figura 4.16: Coordenadas: 36.4, -82

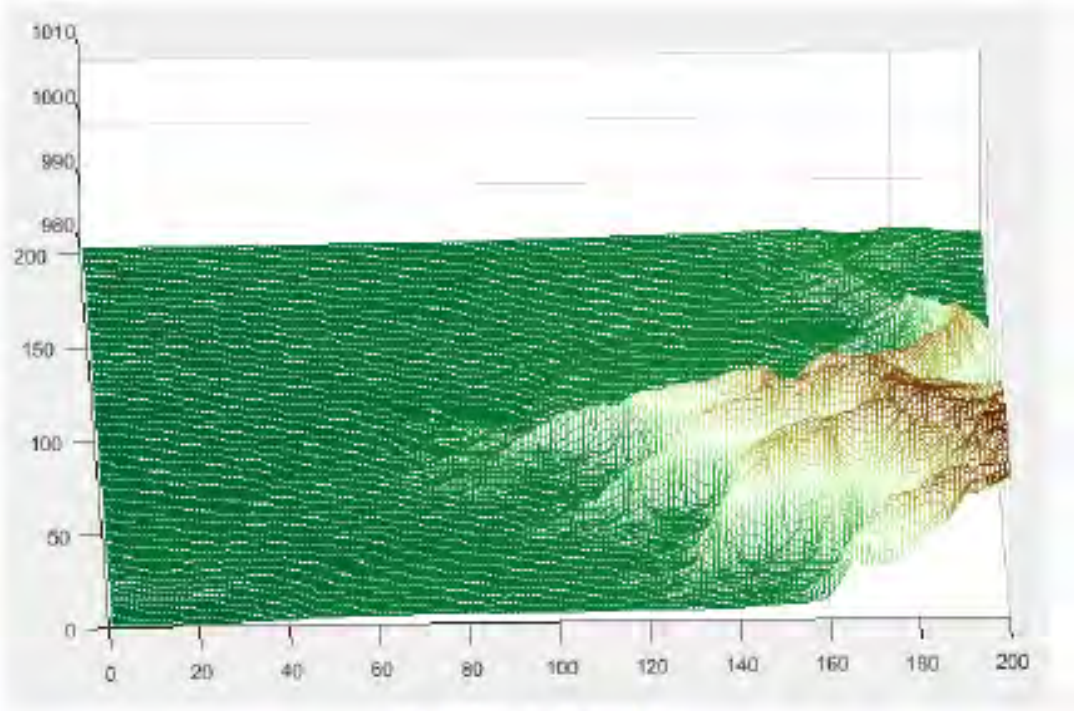


(a) Mapa de elevación

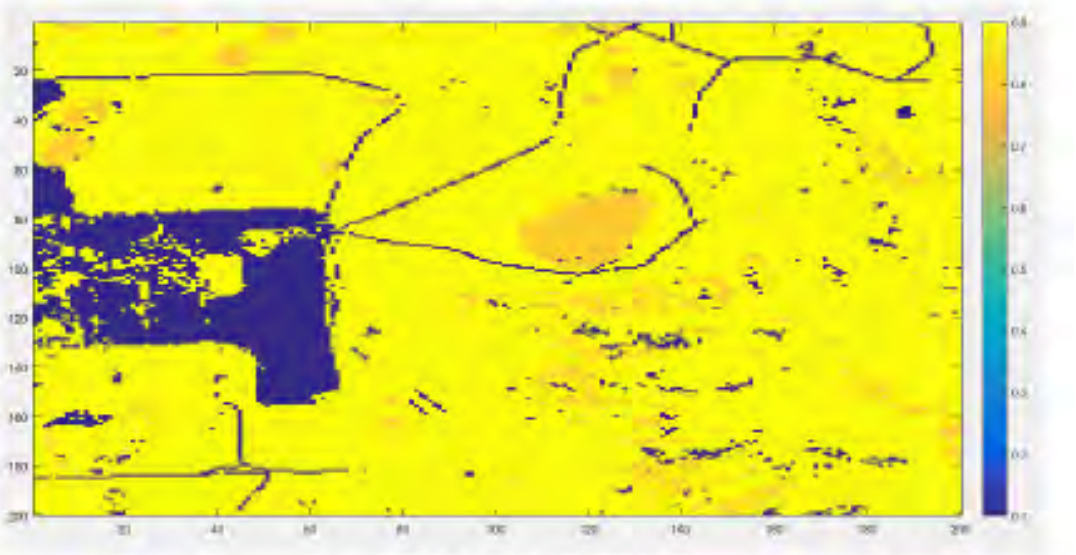


(b) Mapa de velocidad

Figura 4.17: Coordenadas: 40.1, -115.65



(a) Mapa de elevación



(b) Mapa de velocidad

Figura 4.18: Coordenadas: 34.53, -101.2

4.3. Función principal

Pese a que la aplicación está programada como un GUI de Matlab (asunto que se tratará en la sección 4.4), el peso de la preparación de inputs para el uso del algoritmo

y la representación gráfica del resultado de este recae esencialmente en una función de Matlab desarrollada para este TFG: `ejecuta_FM`.

Esta función tiene como *inputs* la matriz de elevación y la matriz de combustible asociada que queramos usar, así como las coordenadas de punto de inicio y final del fuego, valores escalares del viento en la dirección x e y, y el tiempo total que se estima que tardará en propagarse el fuego, frente al punto del tiempo en el cual se quiere ver la representación del fuego.

Todo lo que se pretende hacer en `ejecuta_FM` gira en torno al uso de la función `fast_marching_vectorial_2D`, que equivale a una codificación del algoritmo Fast Marching en C++ (dada la más rápida ejecución que en Matlab). Esta función desarrollada por Gabriel Peyre y mejorada por Santiago Garrido devuelve la matriz de distancias D y la de puntos muertos S del algoritmo FM, para unos puntos iniciales, puntos finales, número de iteraciones máximas, una matriz de costes y dos campos vectoriales, uno en dirección X y otro en Y. De esta manera podemos dividir lo que se hace en `ejecuta_FM` en preparación de los *inputs* para `fast_marching_vectorial_2D` y un tratamiento de la matriz de salida D para la representación del frente sobre el mapa.

En la sección C.4 se encuentra el código completo de `ejecuta_FM` debidamente comentado.

4.3.1. Preparación de *inputs* del algoritmo

Los *inputs* de la función `ejecuta_FM`: puntos de inicio y fin, valores escalares del viento en Y y X y matriz de velocidad tendrán que ser modificados por la función para poder ejecutar el algoritmo. Los cambios más significativos son:

- **Cambio de orientación de la matriz de costes**: La orientación de matrices de Matlab es diferente a la considerada en el algoritmo, por esta razón la matriz de velocidad debe ser reorientada para introducirla en el algoritmo. Se procede primero a una inversión de la matriz con referencia en un eje imaginario horizontal mediante una función llamada `flip_vertical` en el medio de la matriz y después una trasposición de esta.

  De manera análoga, para que una vez conseguida la matriz de propagación D del algoritmo, esta pueda ser representada con las mismas coordenadas que la matriz de elvación, se reorientará D de manera inversa a como se hace con la matriz de velocidad. La figura 4.19 puede servir para entender con mayor claridad el proceso.

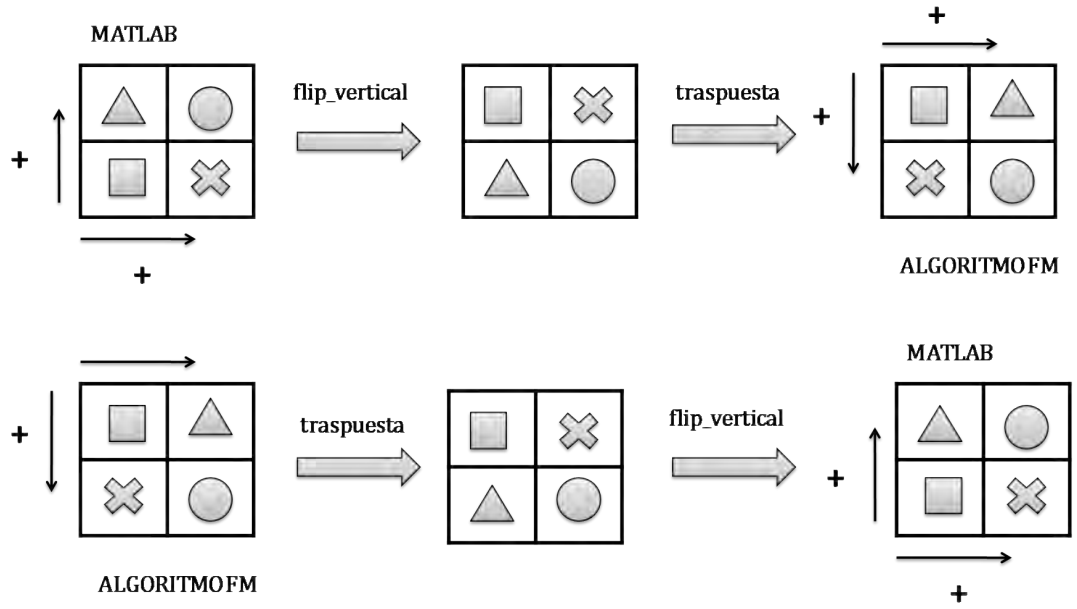


Figura 4.19: Ejemplo de la orientación de ejes en Matlab y el algoritmo y cómo reorientar las matrices

- **Reubicación de los puntos de inicio y fin**: Como consecuencia inmediata del punto anterior se deberán redefinir los puntos iniciales y finales. Las componentes en y de ambos puntos pasarán a ser la dimensión n de la matriz menos el valor en y inicial. Véase la orientación de los ejes en la figura 4.19.

- **Viento como *input* vectorial**: `ejecuta_FM` tiene como *inputs* valores escalares para el viento, ya que partimos del hecho de que este se comporta como un campo vectorial constante en todos los puntos del mapa en cada una de las direcciones x e y. A partir de esos valores se deben diseñar dos matrices VX y VY cuyos valores en todos los puntos son proporcionales a los *inputs* de viento escalares. La creación de estos campos vectoriales constantes es muy sencilla, basta con multiplicar el valor escalar del viento por una matriz unitaria de dimensiones iguales a aquella de la matriz velocidad. El algoritmo FM responde en ocasiones con errores si los campos vectoriales son muy grandes, por tanto para el uso de la función se recomienda no introducir valores escalares del viento de más de 1 dígito. No es difícil percatarse de que esta inclusión del viento como campo vectorial es una prueba, ya que se están considerando valores numéricos aislados con forma de campo vectorial que representan un viento constante. Estos valores no están asociados a ninguna magnitud física de velocidad del viento (km/h, m/s, etc.).

- **Pendiente del terreno como *input* vectorial**: La pendiente del terreno tiene una influencia vectorial como la del viento. Una pendiente positiva en el terreno facilitará la propagación del fuego, ya que este tiende a desplazarse con más facilidad desde niveles de terreno inferiores a niveles de terreno superiores. Esta influencia se

representa de manera simplificada mediante el uso del cálculo del gradiente de la matriz elevacion, obteniendo dos matrices DX y DY con información sobre el gradiente en x e y. La influencia proporcional de este gradiente, sumado a la influencia del viento (matrices VX y VY) constituye el *input* vectorial total del algoritmo FM.

4.3.2. Representación gráfica de resultados

Una vez ejecutado de manera satisfactoria `fast_marching_vectorial_2D` se obtiene la matriz D, que corresponde a la propagación del frente de onda. Pero la representación de este frente de una manera clara sobre el mapa de elevación no es una tarea trivial.

- **Cambio de orientación de la matriz de propagación D**: Así como se transforma la matriz de velocidad para poder ejecutar de manera correcta el algoritmo se debe invertir la operación con D para poder trabajar de manera coherente con ella y el mapa de elevación.
- **Uso de `contour`**: Se trata de una función de Matlab que permite definir y representar gráficamente cuantas lineas de contorno se quiera de un mapa. Cuando se ejecuta `contour` sobre una matriz esta devuelve otra matriz C con todos los puntos pertenecientes a cada línea contorno agrupados en función a la línea a la que pertenezcan.

  Explicado de una manera más clara, C es una matriz con dos filas y un número elevado de columnas, el cual varía en función del número de niveles que se quiera representar. La primera columna contiene información sobre el primer nivel. La primera fila de esta (posición $(1,1)$) indica el nivel al que se encuentra la linea de contorno (para nuestra aplicación es irrelevante) y la segunda indica (posición $(2,1)$) el número de puntos que contiene el nivel. Todas las columnas que siguen a esa primera y hasta llegar a un número de columnas igual al número de puntos en el nivel, contendrán las coordenadas X e Y en las filas 1 y 2 respectivamente. Una vez se llegue al punto con las coordenadas del último punto del nivel, la siguiente columna proporcionará información sobre el siguiente nivel y las columnas siguientes correspondrán a las coordenadas de los puntos de este nivel de manera completamente análoga al primero. En la figura 4.20 se representa un ejemplo del proceso descrito.

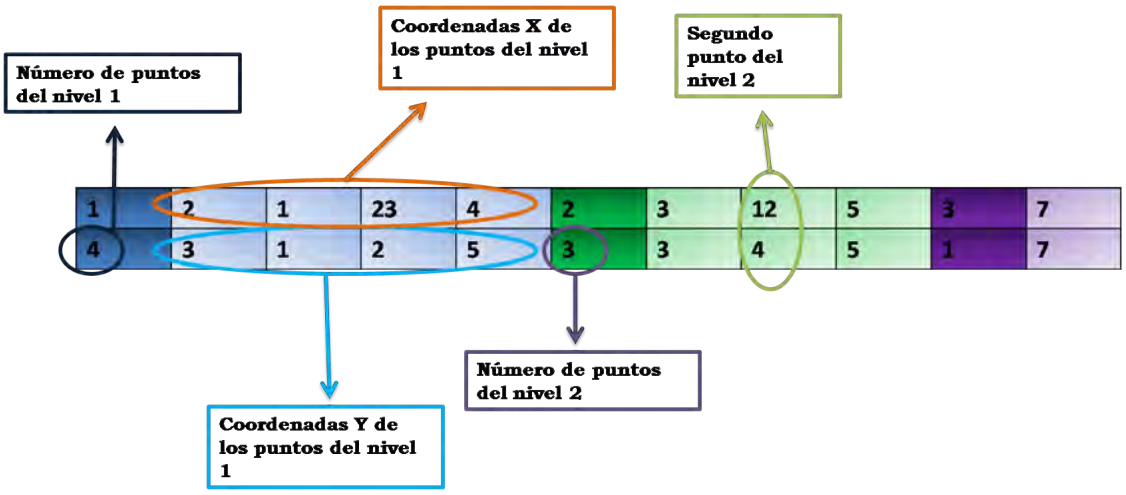


Figura 4.20: Ejemplo sencillo de la estructura de la matriz C

- **Representación de los niveles**: La información en esta matriz proporciona datos suficientes para representar los niveles de contorno en 2D de la matriz D. Pero si se quiere representar el contorno de D sobre la matriz de elevación se deberá almacenar el valor de altura que tiene cada uno de los puntos de los niveles de contorno en la matriz de elevación. El almacenaje de estos puntos se puede hacer en una tercera fila de la matriz C, para mayor comodidad a la hora de programar y a la hora de representar los puntos del frente sobre el mapa de elavación. Ejemplo del proceso en la figura 4.21.

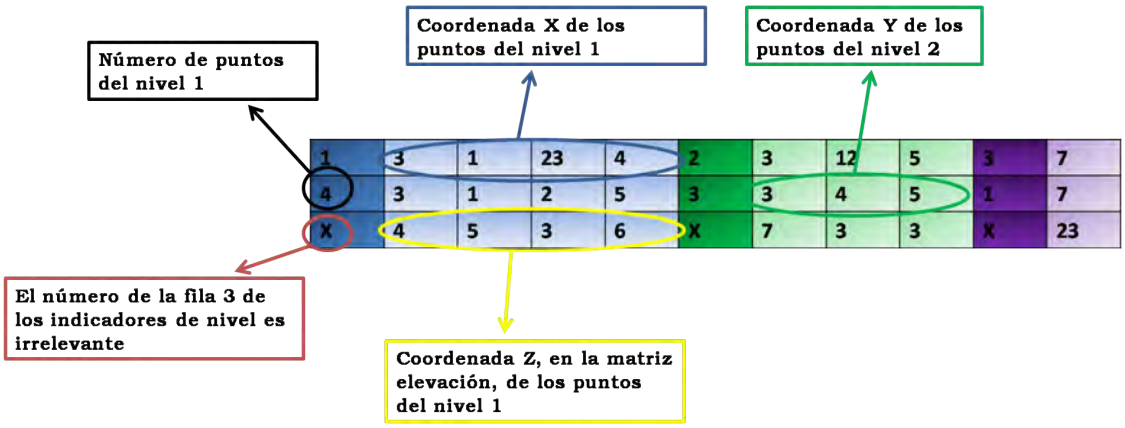


Figura 4.21: Inclusión de la coordenada Z en la matriz C

- **Bucle para la representación de los contornos**: Para que la propagación del fuego se vea como un continuo y no como líneas de contorno aisladas se ha trabajado con muchos niveles de contorno (del orden de 200). Trabajar con tantos niveles de contorno en mapas no sencillos como los mapas de elevación reales que se han utilizado trae pequeños inconvenientes como el hecho de que en realidad existan más niveles de contorno reales que los indicados a la función `contour`, correspondientes

a pequeños círculos en la representación. La representación que se quiere ver encima del mapa de elevación de todos los niveles puede simplificarse con un bucle de tipo `while`.

```
ant=1;
final=longitudC*tiempo/limTiempo;

while ant<final

num=C(2,ant);
cX=C(1, (ant+1):(num+ant));
cY=C(2, (ant+1):(num+ant));
cZ=C(3, (ant+1):(num+ant))+0.02*difAltura*ones(1,length(cY));

plot3(cX,cY,cZ);

ant=ant+1+num;

end
```

Cada iteración del bucle corresponde a las operaciones realizadas para representar gráficamente un nivel de contorno.

Descripción de las variables en el bucle:

- num: Corresponde en cada iteración al número de puntos que contiene cada nivel, se obtiene de la fila 2 de la columna con información de cada nivel.
- ant: Corresponde a la siguiente columna dentro de la matriz C con información sobre el número de puntos del nivel.
- final: Corresponde a la última columna de C que se quiere leer. Es decir se repetirá el bucle, hasta que llegue a ese punto. Esta variable dependerá de la proporción entre las variables tiempo y limTiempo. Si el algoritmo FM ha sido ejecutado para unos puntos de inicio y fin dados, y el fuego tarda un tiempo limTiempo en propagarse, si queremos representar lo que ocurre en un punto intermedio, llamado tiempo, limitaremos la representación hasta un número de columnas de la matriz C:

$$\frac{tiempo}{limTiempo} longitudC \tag{4.1}$$

  Donde longitudC corresponde al número de columnas totales de la matriz C.
- CX,CY y CZ son el conjunto de coordenadas x, y y z respectivamente de todos los puntos de nivel en cada iteración

El conjunto de puntos en cada nivel se puede representar gráficamente mediante la función `plot3`.

En la figura 4.22 para mejor comprensión se presenta un ejemplo de los valores que tomaría cada una de las variables después de la primera iteración.

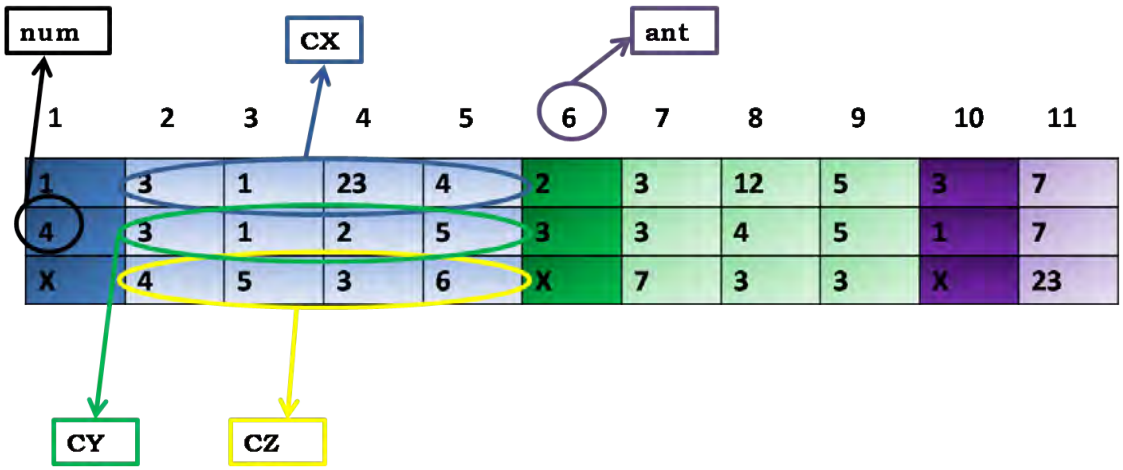


Figura 4.22: Valores que toman las variables tras la primera iteración del bucle en la matriz de ejemplo C

- **Representación en colores del frente**: Se quieren representar todas las lineas de contorno de los diferentes niveles de la matriz de propagación del frente D sobre la matriz de elevación, pero se quiere hacerlo de una manera visualmente atractiva y explicativa. Se necesita que se distingan los niveles iniciales del frente de los más recientes.

  Para hacer todo esto representamos gráficamente el mapa de elevación mediante la función `mesh` de Matlab y utilizamos un mapa de colores muy adecuado para representar terrenos con elevación llamado `demcmap`. Si se representaran las lineas de nivel de D en este mapa tendrían el mismo mapa de color, por lo que para su mejor visualización asignamos un vector con los valores de los colores asociados al mapa de colores `jet`, que representa puntos "pasados" del frente como azules y los puntos más recientes del frente como rojos, bastante adecuado para representar la propagación de un frente. Dado que se necesita un valor de color para cada nivel que se representa, y como se comentó con anterioridad, la matriz C contiene en realidad más niveles de los que nosotros seleccionamos; se precisa utilizar un nuevo bucle para representar gráficamente cada nivel, usando una idea similar a la del bucle para representar los niveles.

```
1 while ant<final
2
3 color=color+1;
4 num=C(2,ant);
```

```
5  ant=ant+1+num;
6
7  end
8
9  j=jet(color);
```

La variable j corresponde a un vector con información sobre los colores a representar y tiene tantos valores como niveles de contorno verdaderamente haya.

4.4. Interfaz gráfica del usuario

La aplicación de simulación de incendios forestales ha sido desarrollada en la interfaz gráfica de usuario de Matlab, GUI. Como cualquier otra interfaz gráfica de cualquier otro programa, un GUI de Matlab permite un control sencillo de las aplicaciones de software para un usuario externo sin necesidad de que este tenga conocimientos de programación.

La programación de un GUI en Matlab resulta más sencilla gracias a `guide`, un entorno de diseño de GUI mediante plantillas.

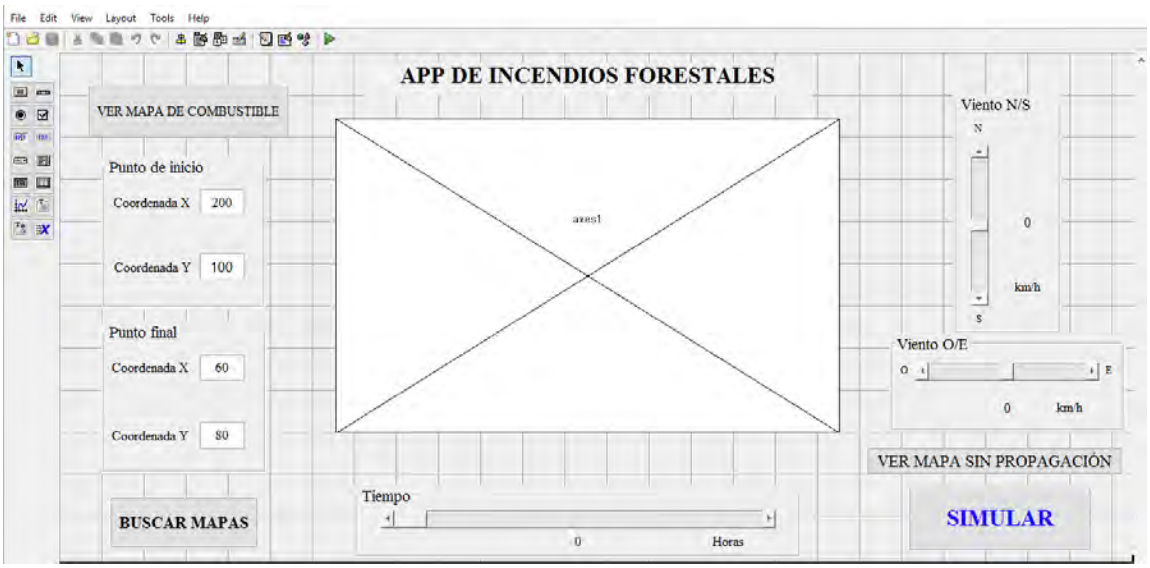


Figura 4.23: Plantilla guide de la aplicación

La plantilla del `guide` permite el uso de diferentes elementos como ejes para representar gráficos o mapas, cuadros de texto, tanto editables como no, *sliders* o botones. Las cualidades de muchos de estos elementos están asociadas a variables de programación. Cuando la plantilla del `guide` está lista, Matlab proporciona un script que al ser ejecutado funciona como la aplicación. En ese código del sript se puede programar a partir de o sobre las variables asociadas a los diferentes elementos de la plantilla.

El código del GUI consta de un conjunto de funciones: una *opening function* , que se ejecuta cada vez que hay algún tipo de interacción del usuario con el GUI, y tantas funciones como elementos de interacción haya en el GUI. El código que se decida programar dentro de cada función se ejecutará solamente en el caso de que el usuario interactúe con esa parte del GUI, por ejemplo, el código dentro de la función de un *pushbutton* (botón de pulsar) se ejecutará cuando el usuario lo pulse. Como en cualquier otra función de Matlab, las variables creadas y utilizadas en cada una de las funciones son locales y mantienen su valor únicamente dentro de la función. Sin embargo, dado que es muy común querer utilizar variables de una función a otra el GUI cuenta con la estructura *handles*, en la que se almacenan variables de interés y que pueden ser utilizadas en cualquiera de las funciones, sin necesidad de recurrir a la programación con variables globales, lo cual no es deseable en programación. Para obtener los valores almacenados en la estructura *handles* se utiliza la función `get`.

A continuación se presenta una explicación más exhaustiva del funcionamiento y lo programado en cada uno de los elementos del GUI:

4.4.1. Elementos del GUI

- ***Sliders* de viento**: Existen dos, uno para el viento N/S (dirección y del gráfico) y otro para el viento O/E (dirección x del gráfico). Orientados de manera que se haga relativamente intuitivo a qué dirección corresponde cada cual.

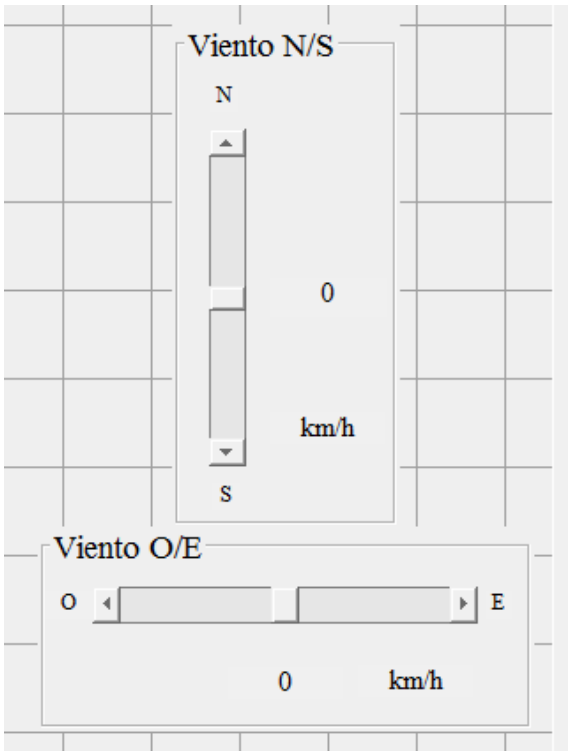


Figura 4.24: *sliders* de viento

Mediante la estructura *handles* se pueden delimitar el máximo y mínimo de los dos sliders: 5 y -5 respectivamente para los dos *sliders*. La elección de estos valores viene dado por el hecho de que el algoritmo no responde siempre bien a campos vectoriales con valores muy altos. Los valores positivos corresponderán a un campo vectorial en el sentido positivo del eje y los negativos a uno en el sentido negativo del mismo.

El valor que toma el *slider* se representa en el texto no editable obteniendo el valor del slider de la estructura *handles* y redondeándolo al entero más cercano y representándolo.

El valor redondeado del *slider* se guarda en la estructura *handles* para que pueda ser usada en `ejecuta_FM`.

- **Coordenadas de inicio y fin**: Representados a partir de cuadros de texto editables. Inicialmente aparecen con unos valores por defecto y se pueden editar.

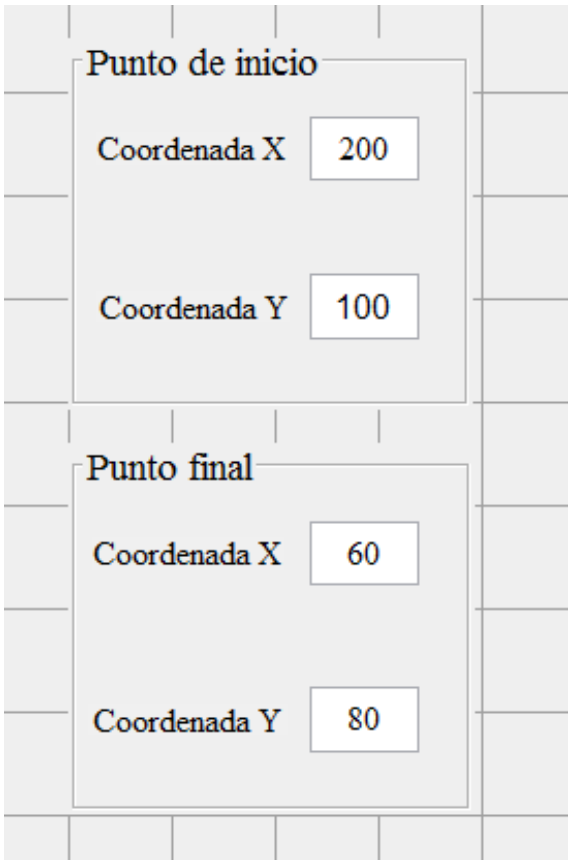


Figura 4.25: Coordenadas, con los valores que aparecen por defecto

La aplicación no permite que se escriba cualquier valor en los campos de las coordenadas. En caso de que el número que se escriba sea mayor que las dimensiones del mapa seleccionado aparece un cuadro de dialogo que informa de que el valor no es admitido (figura 4.26).

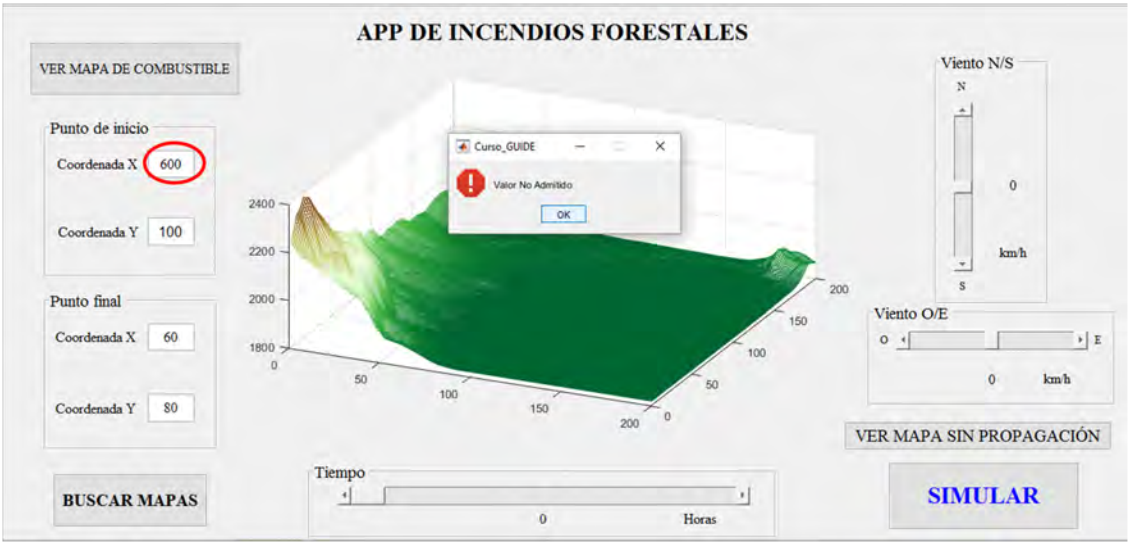


Figura 4.26: Cuadro de diálogo cuando un valor supera las dimensiones de la matriz

- **Botón de búsqueda de mapas**: En la función asociada a este botón se utilizan las funciones `uigetfile` para obtener el *path* del archivo .mat con las matrices de elevación y velocidad y `load` para cargarla en el *workspace* de la función. Una vez cargadas se pasan a la estructura *handles* para poder ser utilizadas en otras funciones del GUI. Cabe recalcar, que dada la manera en la que está programada el GUI es muy importante que las matrices que se importen se llamen respectivamente elevacion y velocidad, ya que en caso contrario el GUI no las podrá almacenar.

- **Botón de visualización del mapa sin propagación**: Se obtiene de la estructura *handles* la matriz elevacion y se representa gráficamente en los ejes del GUI con la función `mesh` y el colormap `demcmap`.

- **Botón de representación del mapa de velocidades**: Mediante el uso del comando `figure` se consigue que la representación gráfica en una ventana a parte, lo que permite la visualización de la propagación del fuego en la gráfica del GUI y el mapa de velocidades al mismo tiempo.

  Para la representación del mapa de velocidades en 2D (ya que se aprecia mejor) se usa la función `imagesc` de Matlab. Dado que esta función representa el sentido positivo del eje y de manera inversa a la función `mesh`, se utiliza la función `flip_vertical` para reorientar la matriz y poder comparar los mapas de elevación y velocidad de manera visual.

- **Botón de simular**: Obtiene de la estructura *handles* los datos necesarios para usar la función `ejecuta_FM`, prepara los ejes del gráfico del GUI y corre la función `ejecuta_FM` con los *inputs* obtenidos del GUI que han sido seleccionados por el usuario.

- **Herramientas del gráfico**: Para permitir al usuario una mejor visualización del mapa, es decir, la capacidad de hacer zoom, alejar y rotar la representación en 3D de los resultados, la aplicación cuenta con iconos en la parte superior izquierda. Estos iconos se pueden añadir en el guide a partir del toolbar editor.

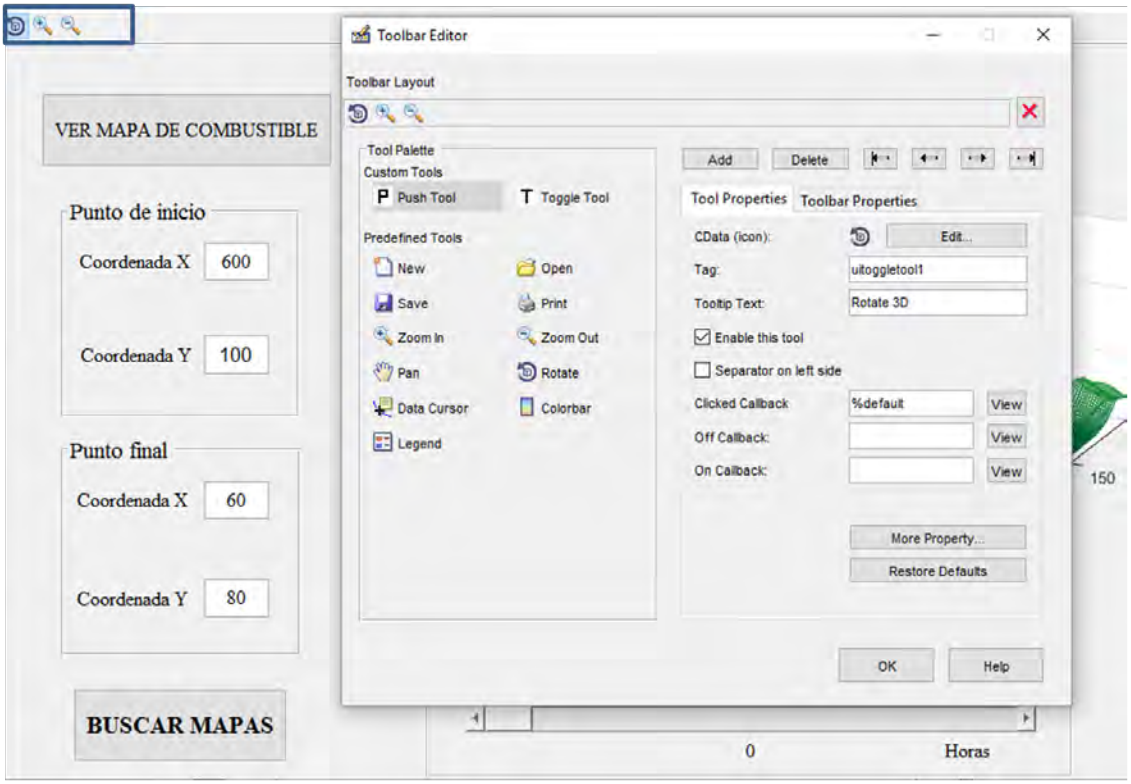


Figura 4.27: Herramientas de la gráfica del GUI

- ***Slider* del tiempo**: El usuario cuenta con un *slider* con el cual puede elegir el momento de tiempo de la propagación del fuego para cada recorrido del fuego para cada par de puntos de inicio y final.

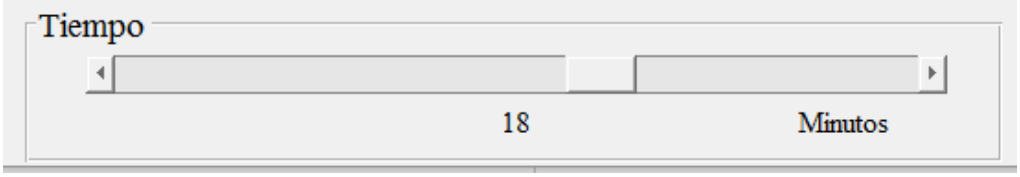


Figura 4.28: Slider del tiempo

El aspecto del tiempo es el más delicado y menos preciso de la aplicación. De momento no es posible calcular de manera exacta y sencilla lo que tardará el fuego en llegar del punto de inicio al punto final. Pero se pretende evitar trabajar con valores completamente aleatorios en el slider y caer en verdaderas faltas de rigor (como estimar que el fuego tardará lo mismo en recorrer la totalidad del mapa a

una pequeña parte de él). Con este fin se han hecho una serie de aproximaciones para el cálculo del valor máximo del slider de tiempo (tiempo que tarda el fuego en ir del punto de inicio al punto final).

Para el cálculo de este valor máximo del *slider*, que llamaremos limTiempo se ha desarrollado una función `calculateLimTiempo`(código en C.3) que a partir de la matriz de velocidades, puntos de inicio y fin y valores escalares del viento en direcciones x e y devuelve el valor aproximado de lo que tarda el fuego en propagarse en esas condiciones. Los criterios de aproximación para la función se explican a continuación:

Para calcular el tiempo estimado de propagación se hará una aproximación de la distancia recorrida y una aproximación de la velocidad media a la que se propaga el fuego.

- **Distancia**: Los mapas que se han descargado en la página web del gobierno americano, Landfire, tienen una resolución de 30m. Es decir, cada celda de la matriz de matlab equivale a un cuadrado de terreno de 30 x 30 metros. Si se trabaja con un punto de inicio de coordenadas (sX, sY) y uno final de (eX, eY), de una manera aproximada se puede determinar la distancia en km que recorrerá el incendio como:

$$distancia = \sqrt{(sX - eX)^2 + (sY - eY)^2}\frac{30}{1000} \tag{4.2}$$

- **Velocidad**: De datos empíricos generales se puede aproximar la velocidad de un incendio forestal normal, en un terreno estándar, sin la acción del viento entre 4km/h y 6km/h, pudiendo llegar hasta 10km/h en terrenos más secos y combustibles. Partiendo de esta información se puede aproximar la velocidad en las zonas donde el mapa de velocidad tiene valor mínimo (0,1) a 1km/h y donde toma valor máximo (0.9) a 9km/h y para valores intermedios hacer una interpolación. Ya que durante todo el camino que recorre el fuego, el mapa de velocidades no suele tener un valor constante se procede a calcular un valor medio en el área que se muestra en la figura 4.29.

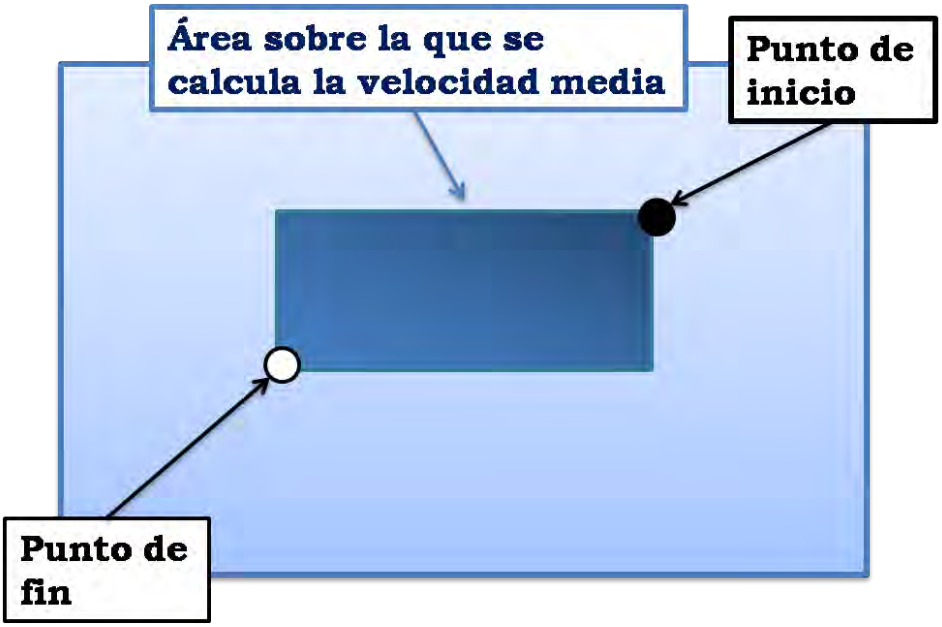


Figura 4.29: Área de la matriz velocidad sobre la cual se calcula el valor medio

Todo esto equivale a una aproximación si no se tiene en cuenta la influencia del viento. Es difícil incluir esta influencia, ya que como se ha comentado anteriormente el viento no tiene influencia física dentro del algoritmo, ya que no puede ser representado en unidades de velocidad (km/h, m/s, etc.). De cualquier manera, aunque no sea muy exacto a nivel físico, para la aproximación del tiempo, se ausumirá que los valores seleccionados por el usuario en los *sliders* del viento estuvieran en km/h. De esa manera, en caso de haber viento favorable hacia el punto final se le sumará la velocidad del viento a la velocidad que se tenía del fuego, y en caso de no serlo, se le restará esta influencia. En caso de que resulte una velocidad negativa, para evitar problemas en la aplicación, se tomará la velocidad como 1km/h.

A partir de los resultados obtenidos para la velocidad y distancia, el tiempo máximo del slider es fácilmente calculable mediante la expresión:

$$tiempo = \frac{distancia}{velocidad} \tag{4.3}$$

Por la magnitud de los valores del tiempo que se obtienen para las matrices con las que se está trabajando es mejor idea trabajar en minutos en vez de en horas. Por esta razón en la función del GUI destinada al *slider* de tiempo, el *output* de la función `calculateLimTiempo` se multiplica por 60 y se delimita el valor máximo del *slider* a este valor. De esta manera, dependiendo de los valores de coordenadas y viento que se elijan, el rango del *slider* variará. Por esta razón, el usuario debe ser especialmente cuidadoso de devolver el slider a su mínimo valor (0) antes de elegir

un nuevo valor (de esta manera se evita que el valor del GUI instantáneo sea mayor que el máximo en algún momento y la aplicación falle).

Capítulo 5

Resultados y Conclusiones

5.1. Muestra de resultados

A continuación se muestran imágenes como representación de los resultados de la aplicación.
Para representar todas las funciones de la aplicación, esta se utilizará con dos pares de mapas de elvación y velocidad. En cada uno de ellos se representarán los resutados de la simulación para diferentes puntos de inicio y fin. Adicionalmente para unos puntos de inicio y fin fijos se variará el tiempo de visualización. Finalmente, para un tiempo de visualización fijo se cambiará la intensidad y dirección del viento.
Al mantener todos los parámetros de la simulación fijos salvo uno se permite al lector una apreciación de la influencia que cada uno de estos parámetros tiene sobre la propagación del fuego y de esa manera poder predecir de manera más exacta el tipo de recorrido que tendrá el fuego.

5.1.1. Primer mapa

Diferentes series de puntos de inicio y fin

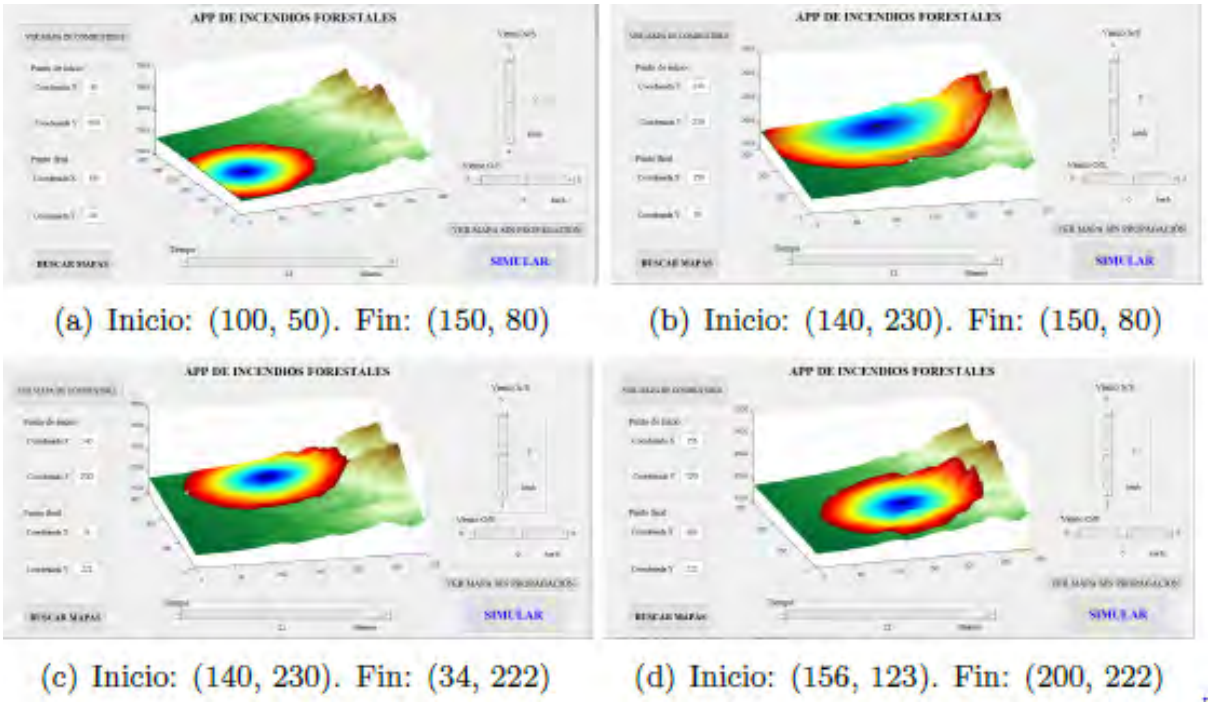
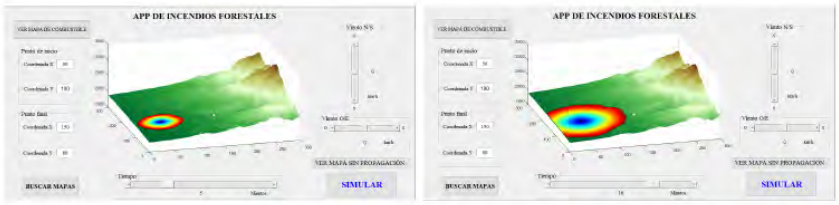
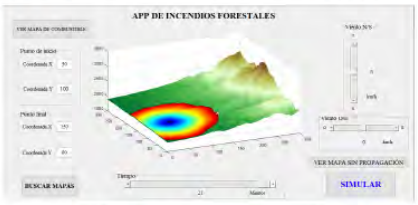


Figura 5.1: Resultado de la simulación para diferentes puntos de inicio y fin

Diferentes tiempos de visualización



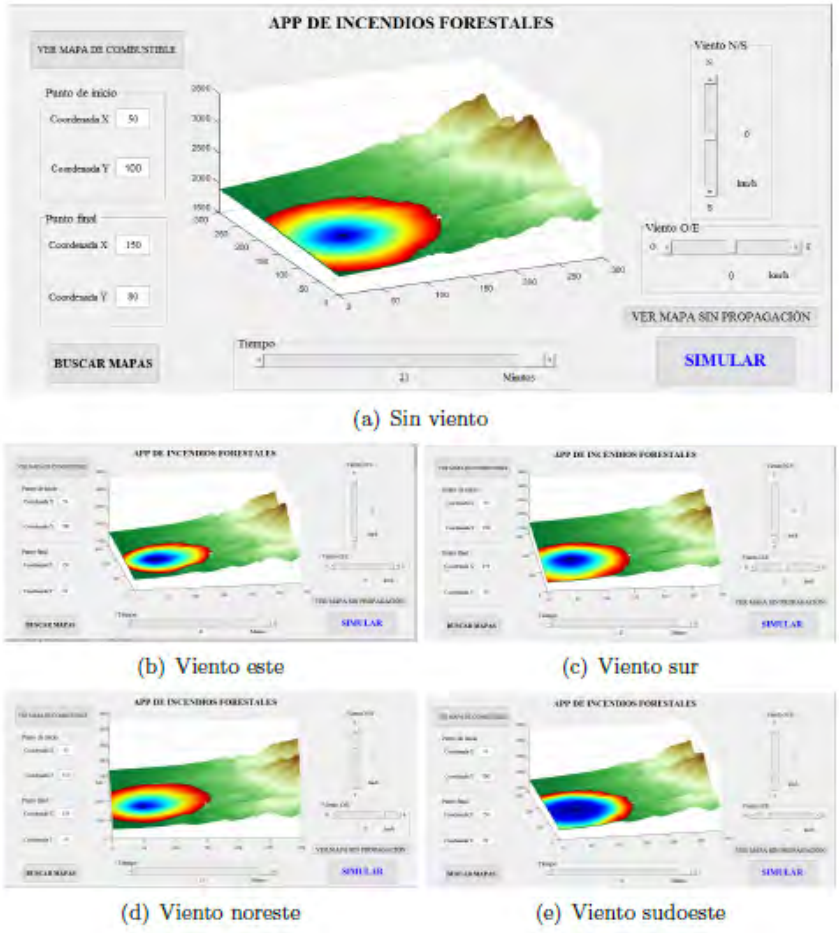
(a) A los 5 minutos (b) A los 16 minutos



(c) A los 21 minutos (tiempo total)

Figura 5.2: Resultado de la simulación para diferentes tiempos

Diferentes valores de viento



(a) Sin viento

(b) Viento este

(c) Viento sur

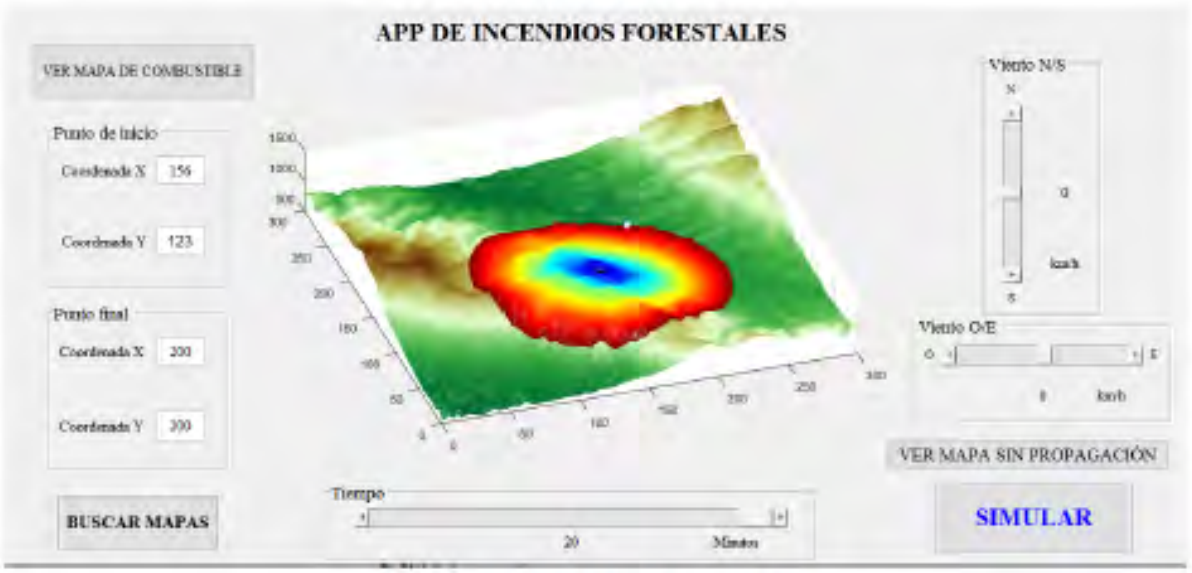
(d) Viento noreste

(e) Viento sudoeste

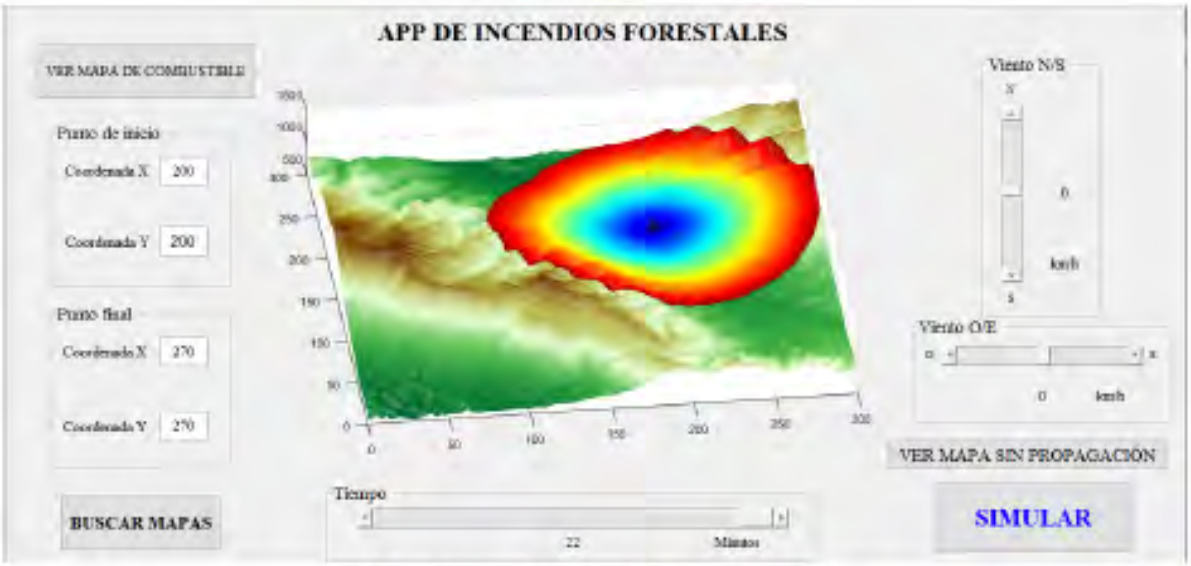
Figura 5.3: Resultado de la simulación para diferentes vientos

5.1.2. Segundo mapa

Diferentes series de puntos de inicio y fin



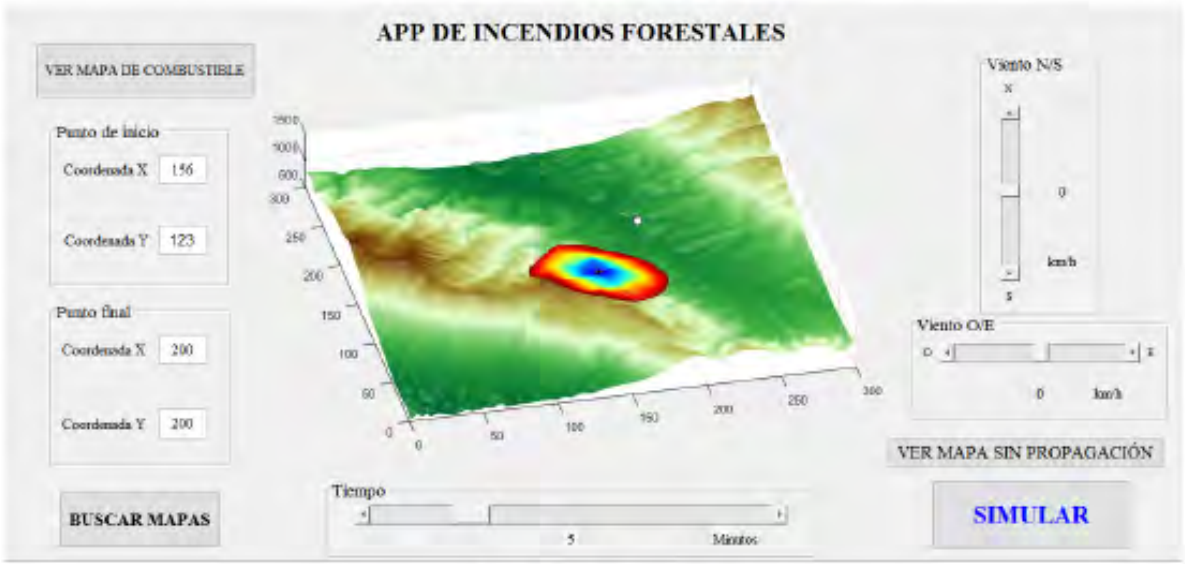
(a) Inicio: (156, 123). Fin: (200,200)



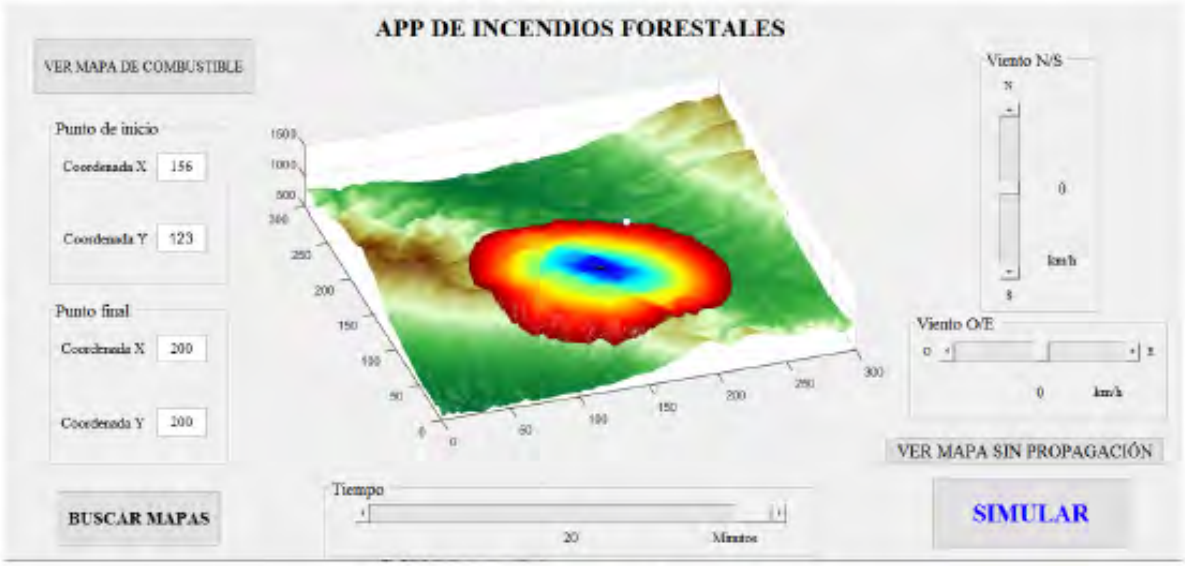
(b) Inicio: (200, 200). Fin: (270,270)

Figura 5.4: Resultado de la simulación para diferentes puntos de inicio y fin

Diferentes tiempos de visualización



(a) A los 5 minutos



(b) A los 20 minutos(tiempo total)

Figura 5.5: Resultado de la simulación para diferentes tiempos

Diferentes valores de viento

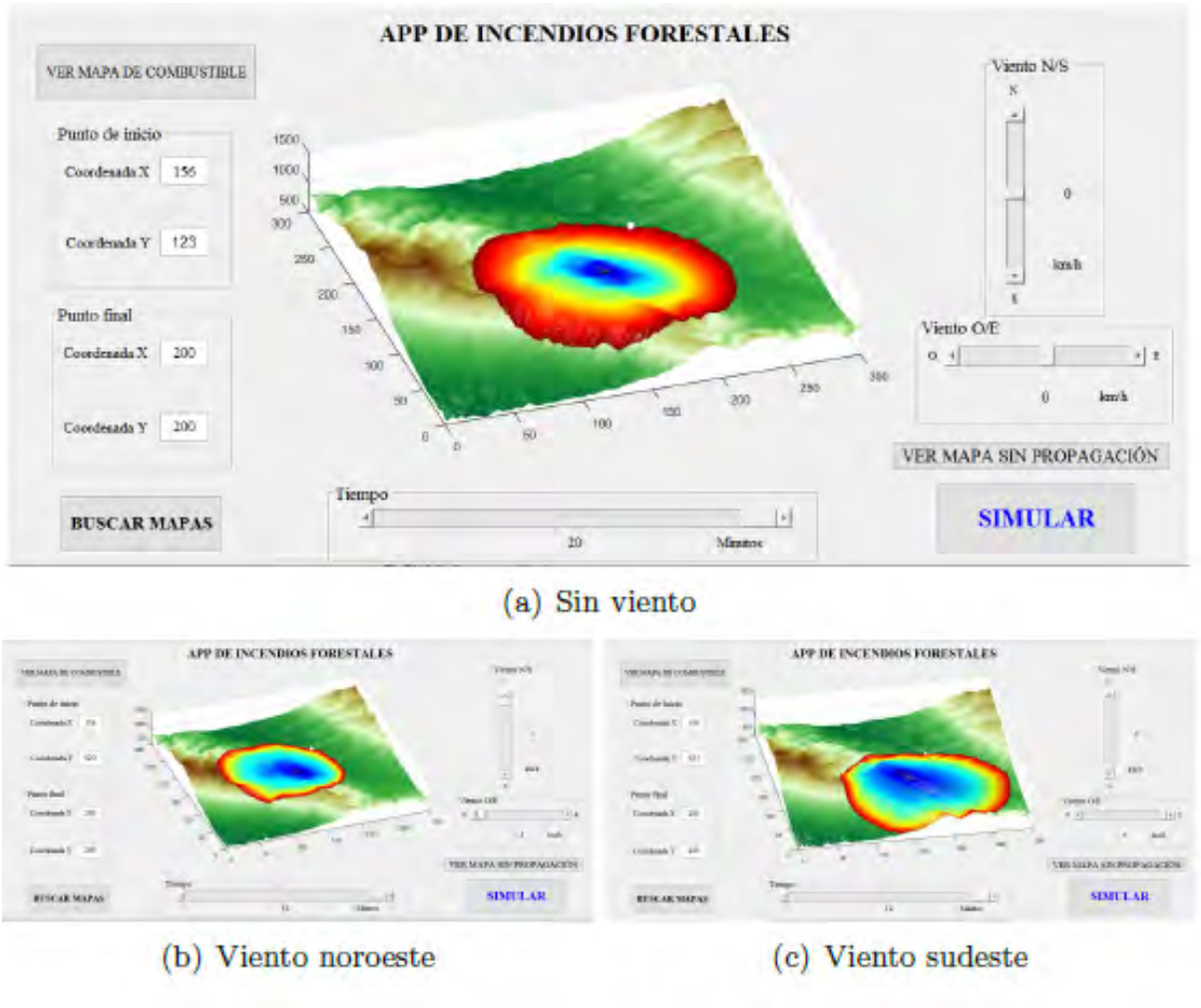


Figura 5.6: Resultado de la simulación para diferentes vientos

5.2. Análisis crítico de los resultados

Una vez desarrollada la aplicación se puede realizar un balance de las ventajas e inconvenientes de esta.

- **Ventajas**: Por una parte, como se puede observar en las imágenes de la sección 5.1, el punto fuerte del proyecto es la representación de resultados y la facilidad de uso de la aplicación. La propagación del fuego es fácilmente visualizable y localizable dentro del mapa y los controles de las variables y su influencia sobre el mapa son lo suficientemente sencillos para que incluso una persona sin conocimientos técnicos pueda utilizarlo con muy pocas explicaciones. Otro punto muy a favor de

la aplicación es la rapidez de computación y representación de resultados. Como se explicó en la sección 3, FM es un algoritmo muy eficiente, y si esto se combina con una programación eficiente del resto la aplicación (como se ha hecho en este TFG) se obtiene una interfaz que muestra los resultados de propagación del incendio en cuestión de segundos de cualquier cambio de las variables que afectan al fuego. Otra de las ventajas es el hecho ante que la aplicación pueda ser utilizada desde el programa Matlab y no necesite un software adicional el cual tenga que ser instalado.

- **Inconvenientes**: El principal inconveniente de esta aplicación viene asociado al hecho de que las unidades, tanto de velocidad del viento como el tiempo transcurridas son muy aproximadas. Otro inconveniente es que, debido al uso exclusivo del algoritmo FM en la aplicación no se puede contemplar el autoapagado del fuego o el retroceso de este en la propagación (hechos que sí se pueden dar en un incendio real).

5.3. Comparación con simuladores existentes

Es difícil realizar una comparación justa entre los simuladores existentes señalados en la sección 2.5 ya que estos se tratan de proyectos muy completos, desarrollados por varias personas y en ocasiones varias entidades, mientras que la aplicación desarrollada en este TFG se trata de una prueba de viabilidad para un proyecto futuro. Por tanto, si bien a nivel de complejidad, uso de recursos y datos esta aplicación no puede competir con grandes infraestructuras como FARSITE o Prometheus, sí tiene varias ventajas con respecto a estos que es relevante señalar:

- **Mayor eficiencia de FM que Huygens**: Como se explicó en la sección 2.5.3 tanto Prometheus como FARSITE utilizan el principio de Huygens para la computación del frente de onda. El cálculo por este método es computacionalmente mucho más lento e ineficiente que el algoritmo FM, debido al coste computacional de tener que calcular cada propagación de onda individual de los puntos del frente y la envolvente de estos.

- **Representación más visual y controles más sencillos**: En la sección 2.5.4 se explicaba la claridad de representación de resultados era mejorable en el programa FARSITE, ya que la linea de propagación de fuego no se distinguía de aquellas del mapa de base (figura 2.8). Así mismo, la utilización del programa para un usuario externo es complicada, ya que se requiere de la completa comprensión del diagrama de la de uso de la figura 2.7. En estos dos aspectos la aplicación desarrollada en este TFG es superior, ya que la propagación del incendio es muy visual y el uso bastante más intuitivo.

5.4. Conclusión

El objetivo de este Trabajo de Fin de Grado era probar la viabilidad del uso de FM para la creación de una aplicación de usuario en Matlab que predijera el desarrollo de un incendio en función de variables como el viento, puntos de inicio y final y mapas de elevación y combustible. En función de si los resultados eran viables esto podría abrir camino para el desarrollo de un proyecto en el departamento de Sistemas y Automática en la Universidad Carlos III de Madrid para un programa a capaz de competir con programas como FARSITE o Prometheus y que suponga una mejora significativa a la hora de luchar contra los incendios forestales.

Vista la posibilidad de desarrollar una aplicación y las ventajas que esta, aún siendo sencilla y pequeña, tiene sobre programas más desarrollados ya existentes, parece seguro decir que es viable la implementación de un programa de predicción de incendios que use el algoritmo Fast Marching.

Capítulo 6

Mejoras y Trabajo Futuro

Con vistas a la elaboración de un proyecto de investigación para el desarrollo de programa que ayude a simular incendios forestales basado en el trabajo de este TFG, se presentan a continuación una serie de mejoras a realizar sobre este:

- **Unidades**: Probablemente el aspecto más importante a mejorar en la aplicación sea conseguir la coherencia entre la representación del frente y los valores del viento y tiempo. Es decir, que los valores de viento y tiempo que el usuario elija, con sus unidades (km/h para velocidad el viento y horas o minutos para el tiempo) se traduzcan de manera coherente en los metros que recorre el frente de propagación del fuego. Como se ha comentado en secciones anteriores no se trata de una cuestión trivial en absoluto ya que el algoritmo FM al ser un algoritmo númerico en vez de físico o empírico no trabaja con unidades fijas. En cualquier caso existe un amplio campo de trabajo para, mediante aproximaciones, conseguir una estimación de las magnitudes físicas y una buena correlación con los resultados del algoritmo. Mediante estudios empíricos, comparando casos reales con lo que hace la aplicación para diferentes *inputs* de viento y tiempo, se pueden llegar a conclusiones bastante precisas que pueden ser extrapoladas a otros casos. Por ejemplo, en el caso de un incendio ya sucedido, sobre el que hay datos, conociendo el viento que sopló ese día, se pueden estudiar los valores de viento que hacen que el incendio se comporte en la aplicación como se comportó en la realidad y a partir de ahí extrapolar unidades para un caso general. Además se podría continuar con lo realizado en la línea de lo hecho en la sección 4.4.1, pero de una manera mucho más exacta: calculando de manera más minuciosas distancia y velocidad estimada de propagación del fuego. Tal vez pudiera extrapolarse esta aproximación para conseguir también unos valores del viento con unidades físicas.

- **Algoritmo**: Una mejora en la implementación del algoritmo en Matlab/C++ (`fast_marching_vectorial_2D`), puede suponer una mejora de la calidad de la aplicación muy significativa. Pese a no ser competencia directa del TFG (se recuerda que la aplicación trabaja con un código ya proporcionado) tendría grandes ventajas en la aplicación. Por ejemplo, si pudiera ofrecerse una solución al hecho de que la aplicación falla cuando el frente de fuego intenta retroceder implicaría una serie de mejoras bastante significativas. Es evidente que por naturaleza el algoritmo FM no puede retroceder, pero si se encontrara una manera alternativa al algoritmo para esos casos, supondría una mejora significativa. Se podría trabajar con matrices de mayores dimensiones en la aplicación así como con mapas de velocidad sin necesidad de ser tratados de manera tan exhaustiva como se hace para que la aplicación funcione de manera correcta.

- **Más seguridad de la aplicación frente al usuario**: Pese a que la aplicación es relativamente intuitiva de utilizar, si se pretende que la puedan utilizar personas que no necesariamente sepan de programación o de ingeniería en general se deberían mejorar más ciertos detalles de seguridad de la aplicación. Por ejemplo, conseguir que la aplicación pueda trabajar con mapas de velocidad y elevación en cualquier

formato, es decir, que la transformación a archivos .mat se hiciera dentro de la propia aplicación. O introducir más avisos de error dentro de la aplicación similares al que emite cuando las coordenadas que el usuario decide están fuera de los límites de la matriz. También podrá incluirse un botón de ayuda para guiar al usuario en el uso de la aplicación.

- **Estética de la aplicación**: Hay cierto margen de mejora en la estética de la aplicación también para tratar de que sea más estructurada a la vista.

- **Inclusión de nuevos parámetros**: Se podría incluir una manera de que el usuario subiese a la aplicación información adicional de carácter meteorológico. O por ejemplo modificar el *slider* del tiempo para poder trabajar con tiempos reales (establecer una hora del día en concreto para el inicio del fuego). Adicionalmente se podría incluir información dentro del frente de propagación sobre el tiempo (en determinadas líneas de la propagación del frente incluir a qué hora corresponde cada una).

- **Optimización en la programación**: La aplicación desarrollada no es especialmente lenta, y si lo es, es en general por temas de representación gráfica en Matlab. Aún así se trata de una aplicación no desarrollada por un programador experto. Mediante una revisión del código con intención de optimizar los diferentes bucles probablemente se podría reducir el tiempo de ejecución hasta conseguir una actualización inmediata de la visualización del frente con cualquier cambio en cualquiera de los parámetros.

- **Viento**: El viento supone un campo de mejora en la aplicación bastante amplio. En la aplicación se utiliza una aproximación un tanto arriesgada en comparación con lo que ocurre en realidad, ya que el viento no se comporta como un campo constante vectorial en 2D (cuya dirección depende de los valores escalares que se de a sus componentes x e y). Si se quisiera una representación más fiable de lo que ocurre en la realidad debería tenerse en cuenta que el viento no tiene porqué comportarse de manera constante en todos los puntos (existen ráfagas, remolinos, etc.). Otro punto muy importante es el hecho de que las simulaciones se hacen sobre terrenos montañosos, por esa razón el viento, aunque meteorológicamente fuera constante se adaptaría a la superficie montaosa. Una forma de adaptar el viento a la superficie del mapa de elevación es mediante el uso del propio algoritmo FM.

- **Validación de resultados**: Una vez se hayan instaurado las mejoras propuestas anteriormente, para que el proyecto del programa de simulación tuviera una utilidad real se necesitaría una extensa validación de los resultados del programa, contrastando sus resultados para unas condiciones determinadas con incendios reales ocurridos.

Esta serie de propuestas de mejora serán de utilidad para la eficiente continuación del proyecto.

Apéndices

Apéndice A

Marco Regulador

Al tratarse de un proyecto de investigación que pretende el desarrollo de una aplicación que predice la propagación de un incendio, cualquier marco regulador, de carcter legal o económico se aplicaría a acciones destinadas a la extinción de los fuegos basadas en las predicciones del simulador, pero ninguna se aplica al simulador como programa.

Apéndice B

Entorno Socio-Económico

Al tratarse de una prueba de viabilidad del desarrollo de un proyecto futuro, la aplicación desarrollada en este TFG no tiene impacto inmediato más allá que la confirmación de que es viable la inversión de trabajo y recursos en una continuación y mejora del proyecto. Sin embargo, de desarrollarse dicho proyecto este sí tendría un fuerte impacto de carácter medioambiental, al verse reducidas las hectáreas quemadas debido a una mayor rapidez y eficiencia en el apagado de incendios (se conocería qué camino tomará el incendio, por tanto los lugares claves para su apagado, como utilizar las condiciones del terreno o meteorología a favor de los equipos de apagado). De la mano del impacto medioambiental habría también un impacto económico, ya que una extinción más efectiva conllevaría una reducción en gasto en material, operativos y gasto gubernamental para reparar zonas quemadas.

B.1. Presupuesto

En la tabla B.1 se presenta el presupuesto necesario para llevar a cabo este trabajo de Fin de Grado. Por tratarse de una aplicación software que utiliza el programa Matlab, no tendrá costes materiales y todo el presupuesto vendrá determinado por las horas de trabajo así como el coste de la licencia de Matlab.

Descripción	**Unidades**	**Precio unitario**	**Coste**
Licencia académica individual de Matlab	1	500 $\frac{e}{unidad}$	500 €
Horas trabajadas	400 horas	10 $\frac{e}{hora}$	4000 €
Total neto			4500 €
IVA (21 %)			945 €
Total			**5445 €**

Tabla B.1: Presupuesto

Como se puede apreciar, se trata de un proyecto con un coste total muy bajo. Pese a que la implantación de mejoras mencionadas en el capítulo 6 pueden suponer costes adicionales, si se retoma la cifra de gasto en extinción de incendios en España de 1776 millones de euros, mencionada en la sección 1.1, se trata de una solución al problema increíblemente eficiente a nivel ecónomico.

Apéndice C

Códigos

C.1. Algoritmo Dijkstra

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%EJEMPLO SENCILLO DE LA IMPLEMENTACI N DEL ALGORITMO DE DIJKSTRA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Tenemos una malla de 3x3 para que sea sencillo

n=3;
%Contiene los vectores de desplazamiento en cualquiera de las 4 ...
    direcciones
%de la malla

neigh = [[1;0] [-1;0] [0;1] [0;-1]];

%Establece la fila en la que se encuentra el punto que se elija

boundary = @(x)mod(x-1,n)+1;

%Las siguientes funciones hacen posible que dado un punto de la ...
    malla, k,
%se mueva en dirección i, siendo i cada uno de los vectores de ...
    dirección.

ind2sub1 = @(k)[rem(k-1, n)+1; (k - rem(k-1, n) - 1)/n + 1];
sub2ind1 = @(u)(u(2)-1)*n+u(1);
Neigh = @(k,i)sub2ind1( boundary(ind2sub1(k)+neigh(:,i)) );

%Hacemos que el mapa de costes tenga valor 1 para todos los nodos

W = ones(n);

%Establecemos el punto medio de la malla como punto de inicio

```

```
30 x0 = [n/2;n/2];
31
32 %Inicialización de D, con todos puntos desconocidos (distancia infinito)
33 %salvando el punto inicial.
34
35 I = sub2ind1(x0);
36 D = zeros(n)+Inf;
37 D(I) = 0;
38
39 %De manera análoga se inicializa la matriz con información sobre el ...
      estado
40 %de los puntos. Todos 0 salvando el inicial que vale 1
41
42 S = zeros(n);
43 S(I) = 1;
44
45 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
46 %Comienza la parte iterativa
47 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
48
49 %Actualización para que "mueran" lo estados de los puntos ya ...
      considerados
50
51 [tmp,j] = sort(D(I)); j = j(1);
52 i = I(j); I(j) = [];
53 S(i) = -1;
54
55 %Consideramos los puntos "vecinos", descartando aquellos que ya estén
56 %"muertos"
57
58 J = [Neigh(i,1); Neigh(i,2); Neigh(i,3); Neigh(i,4)];
59 J(S(J)==-1) = [];
60
61 %Añadimos los "vecinos" que no hayan sido considerados con ...
      anterioridad a S
62
63 J1 = J(S(J)==0);
64 I = [I; J1];
65 S(J1) = 1;
66
67 %Efectuamos la actualización del valor de las distancias en la ...
      matriz D
68
69 for j=J'
70     dx = min( D([Neigh(j,1) Neigh(j,2)]) );
71     dy = min( D([Neigh(j,3) Neigh(j,4)]) );
72     D(j) = min(dx+W(j), dy+W(j));
73 end
```

C.2. Algoritmo Fast Marching

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%EJEMPLO SENCILLO DE LA IMPLEMENTACI N DEL ALGORITMO FMM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Tenemos una malla de 3x3 para que sea sencillo

n=3;

%Contiene los vectores de desplazamiento en cualquiera de las 4 ...
   direcciones
%de la malla

neigh = [[1;0] [-1;0] [0;1] [0;-1]];

%Establece la fila en la que se encuentra el punto que se elija

boundary = @(x)mod(x-1,n)+1;

%Las siguientes funciones hacen posible que dado un punto de la ...
   malla, k,
%se mueva en dirección i, siendo i cada uno de los vectores de ...
   dirección.

ind2sub1 = @(k)[rem(k-1, n)+1; (k - rem(k-1, n) - 1)/n + 1];
sub2ind1 = @(u)(u(2)-1)*n+u(1);
Neigh = @(k,i)sub2ind1( boundary(ind2sub1(k)+neigh(:,i)) );

%Hacemos que el mapa de costes tenga valor 1 para todos los nodos

W = ones(n);

%Establecemos el punto medio de la malla como punto de inicio

x0 = [n/2;n/2];

%Inicialización de D, con todos puntos desconocidos (distancia infinito)
%salvando el punto inicial.

I = sub2ind1(x0);
D = zeros(n)+Inf;
D(I) = 0;

%De manera análoga se inicializa la matriz con información sobre el ...
   estado
%de los puntos. Todos 0 salvando el inicial que vale 1

S = zeros(n);
S(I) = 1;

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Comienza la parte iterativa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Actualización para que "mueran" lo estados de los puntos ya ...
    considerados

[tmp,j] = sort(D(I)); j = j(1);
i = I(j); I(j) = [];
S(i) = -1;

%Consideramos los puntos "vecinos", descartando aquellos que ya estén
%"muertos"

J = [Neigh(i,1); Neigh(i,2); Neigh(i,3); Neigh(i,4)];
J(S(J)==-1) = [];

%Añadimos los "vecinos" que no hayan sido considerados con ...
    anterioridad a S

J1 = J(S(J)==0);
I = [I; J1];
S(J1) = 1;

%Efectuamos la actualización del valor de las distancias en la ...
    matriz D

for j=J'
    dx = min( D([Neigh(j,1) Neigh(j,2)]) );
    dy = min( D([Neigh(j,3) Neigh(j,4)]) );
    Delta = 2*W(j) - (dx-dy)^2;
if abs(dx-dy)<=W(j)
    D(j) = (dx+dy+sqrt(Delta))/2;
else
    D(j) = min(dx+W(j), dy+W(j));
end
end
```

C.3. Función calculateLimTiempo

```
function limTiempo=calculateLimTiempo(velocidad, sX,sY,eX,eY, ...
    vientoX, vientoY)

%Devuelve el tiempo aproximado, limTiempo, en horas que tarda el ...
    fuego en llegar desde el punto de inicio, coordenadas (sX,sY) al ...
    punto final, coordenadas (eX, eY) en el campo de velocidades, ...
    velocidad, con unos valores de viento escalares en las ...
    direcciones x e y respectivamente, vientoX y vientoY

```

```
5  %Cálculo del fragmento del mapa
6  %de velocidades sobre el cuyos valores se calcula la media de velocidad
7
8  cacho=velocidad(min(sX,eX):max(sX,eX), min(sY,eY):max(sY,eY));
9
10 velSV=10*mean(mean(cacho));
11
12 %En las condiciones presentadas a continuación se estudia el efecto del
13 %viento. Dependiendo de la situación de los puntos de inicio y fin a la
14 %velocidad dada proporcionalmente por el mapa de velocidades se le ...
       sumará o
15 %restará el efecto del viento en dirección X o Y
16
17 if eY>sY
18
19     vel=velSV+vientoY;
20
21 elseif eY<sY
22
23     vel=velSV-vientoY;
24
25 else
26
27    vel=velSV;
28
29 end
30
31 if eX>sX
32
33     vel=velSV+vientoX;
34
35 elseif eX<sX
36
37     vel=velSV-vientoX;
38
39 else
40
41    vel=velSV;
42
43 end
44
45 %En el caso de que por los cálculos saliera una velocidad negativa se
46 %tomará como valor 1 para evitar problemas
47
48 if vel<0
49
50     vel=1;
51
52 end
53
54 %Cálculo de la distancia mínima recorrida por el fuego para llegar ...
       al punto
55 %final (km)
```

```
56
57 distancia=sqrt((eX-sX)^2+(eY-sY)^2)*30/1000;
58
59 %Cálculo del tiempo a partir de la distncia y velocidad
60
61 limTiempo=distancia/vel;
62
63 end
```

C.4. Función `ejecuta_FM`

```
1  function ejecuta_FM_prueba(elevacion,velocidad,tiempo, vientoX, ...
       vientoY, sX, sY, eX, eY, limTiempo)
2
3  %Ejecucuta el algoritmo FM y adicionalmente representa gráficamente ...
       la propagación del
4  %fuego sobre el mapa de elevación que se le proporcione
5
6  %elevacion es la matriz con los datos de elevación del mapa en cuestión
7  %velocidad es la matriz de coste o velocidad para introducir en el ...
       algortimo FM, tendrá la misma orientación y dimensiones que elevacion
8  %tiempo es el valor de tiempo del slider
9  %vientoX y vientoY son los valores del viento en las direcciones X e ...
       Y respectivamente de los sliders
10 %sX, sY, eX y eY puntos iniciales y finales respectivamente.
11 %limTiempo es el rango máximo que cubre el slider tiempo, que ...
       debería cambiar dentro del GUI en función del valor de los ...
       starting y ending points
12
13 hold off
14
15 [dim1,dim2]=size(elevacion);%dimensiones de la matriz, necesarias ...
       para meter en el algoritmo
16
17 %Rango de alturas con el que se trabaja en la matriz elevacion
18
19 maxAltura=max(max(elevacion));
20 minAltura=min(min(elevacion));
21 difAltura=maxAltura-minAltura;
22
23 %Orientar la matriz para que se pueda ejecutar el algoritmo
24
25 velocidad=flip_vertical(velocidad);
26 velocidad = velocidad';
27
28 %Wo es la matriz de elevación rotada para poder ejecutar el algoritmo.
29
30 Wo1=rescale(elevacion,0.1,0.9);
31 Wo1=flip_vertical(Wo1);
```

```
32 Wo1 = Wo1';
33 Wo=1-Wo1;
34
35 [DX,DY] = gradient(Wo);%efecto de la montaña viene dado por el ...
      gradiente de del mapa de alturas original, no del campo de ...
      velocidades.
36
37 %Campos vectoriales en dirección X e Y
38
39 FX =-0.5*vientoX*ones(dim1)+DX;
40 FY =0.5*vientoY*ones(dim2)+DY;
41 N=ones(dim1,dim2);
42
43 %La imagen sale invertida, y para que la coordena y que se introduce
44 %coincida con la real en al gráfico
45
46 sY1=dim2-sY;
47 eY1=dim2-eY;
48
49 %Se inicializa el punto inicial o foco y se desiigna el punto final
50
51 start_points = [sX;sY1];
52 end_points = [eX;eY1];
53
54 % FMM as usual (options)
55
56 options.nb_iter_max = inf;
57 options.end_points = end_points;
58
59 [D,S] = fast_marching_vectorial_2d(velocidad, start_points, options, ...
      FX,FY, N); %ejecutamos algoritmo FM
60
61 d=flip_vertical(D');%"Desrotamos" la matriz D
62
63 nNiveles=200;%Número de niveles del contorno
64
65 C=contour(d,nNiveles); %Matriz con la información de los contornos ...
      de d. Elegimos cuantos niveles queremos
66
67 %Usaremos la propia matriz C para guardar los valores Z del contorno
68 %(aquellos correspondientes a W
69
70 longitudC=length(C);
71
72 for i=1:longitudC
73
74     if C(2,i)<=dim1 %Solucionar problemas con los indicadores de nivel.
75         %Sí, hallamos las alturas de los indicadores de nivel, pero ...
              computacionalmente cuesta menos que hacerlos desaparecer
76
77         C(3,i)=elevacion(ceil(C(2,i)), ceil(C(1,i)));%Por alguna ...
              extraña razón X e Y tienen que estar invertidos. Ceil ...
              redondea hacia arriba.
```

```
78
79      end
80  end
81
82  %Sacado y ploteado de los niveles sobre la matriz de elevación
83
84  mesh(elevacion); hold on %Ploteamos primero la montaña en 3D
85
86  %Le ponemos el colormap para la montaña
87
88  zlimits = [min(elevacion(:)) max(elevacion(:))];
89  demcmap(zlimits);
90
91  %Primer bucle para sacar los niveles que verdaderamente hay (por alguna
92  %razón en realidad hay más de los establecidos, son pequeños circulitos)
93
94  ant=1;
95  color=0;
96
97  final=longitudC*tiempo/limTiempo;
98
99  while ant<final
100
101 color=color+1;
102 num=C(2,ant);
103 ant=ant+1+num;
104
105 end
106
107 %La variable color tiene el número de verdaderos niveles que hay
108
109 j=jet(color); %Queremos que la propagación de la onda se haga acorde ...
       con los colores del colormap jet.
110 %Sacamos un vector con la información, con tantas celdas como ...
       verdaderos niveles
111
112 %Usamos el loop anterior pero esta vez ploteando cada uno de los niveles
113
114 ant=1;
115 color=0;
116
117 while ant<final
118
119 color=color+1;
120 num=C(2,ant);
121 cX=C(1, (ant+1):(num+ant));
122 cY=C(2, (ant+1):(num+ant));
123 cZ=C(3, ...
       (ant+1):(num+ant))+0.02*difAltura*ones(1,length(cY)); %Sumamos una ...
       pequeña altura para mejor visualización
124
125 plot3(cX,cY,cZ,'Color', j(color,:));
126
```

```
127 ant=ant+1+num;
128
129 end
130
131 %Representación de los puntos de inicio y final
132
133 scatter3(sX,sY,elevacion(sY,sX)+0.025*difAltura,'o','MarkerFaceColor','k');
134 scatter3(eX,eY,elevacion(eY,eX)+0.025*difAltura,'o','MarkerFaceColor','w');
```

C.5. GUI

```
1  function varargout = trasteando(varargin)
2  % TRASTEANDO MATLAB code for trasteando.fig
3  %      TRASTEANDO, by itself, creates a new TRASTEANDO or raises the ...
       existing
4  %      singleton*.
5  %
6  %      H = TRASTEANDO returns the handle to a new TRASTEANDO or the ...
       handle to
7  %      the existing singleton*.
8  %
9  %      TRASTEANDO('CALLBACK',hObject,eventData,handles,...) calls ...
       the local
10 %      function named CALLBACK in TRASTEANDO.M with the given input ...
       arguments.
11 %
12 %      TRASTEANDO('Property','Value',...) creates a new TRASTEANDO ...
       or raises the
13 %      existing singleton*.  Starting from the left, property value ...
       pairs are
14 %      applied to the GUI before trasteando_OpeningFcn gets called.  An
15 %      unrecognized property name or invalid value makes property ...
       application
16 %      stop.  All inputs are passed to trasteando_OpeningFcn via ...
       varargin.
17 %
18 %      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows ...
       only one
19 %      instance to run (singleton)".
20 %
21 % See also: GUIDE, GUIDATA, GUIHANDLES
22
23 % Edit the above text to modify the response to help trasteando
24
25 % Last Modified by GUIDE v2.5 19-May-2017 13:02:15
26
27 % Begin initialization code - DO NOT EDIT
28 gui_Singleton = 1;
29 gui_State = struct('gui_Name',       mfilename, ...
```

```
30                    'gui_Singleton',  gui_Singleton, ...
31                    'gui_OpeningFcn', @trasteando_OpeningFcn, ...
32                    'gui_OutputFcn',  @trasteando_OutputFcn, ...
33                    'gui_LayoutFcn',  [] , ...
34                    'gui_Callback',   []);
35 if nargin && ischar(varargin{1})
36     gui_State.gui_Callback = str2func(varargin{1});
37 end
38
39 if nargout
40     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
41 else
42     gui_mainfcn(gui_State, varargin{:});
43 end
44 % End initialization code - DO NOT EDIT
45
46
47 % --- Executes just before trasteando is made visible.
48 function trasteando_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
50 % hObject    handle to figure
51 % eventdata  reserved - to be defined in a future version of MATLAB
52 % handles    structure with handles and user data (see GUIDATA)
53 % varargin   command line arguments to trasteando (see VARARGIN)
54
55
56
57 % Choose default command line output for trasteando
58 handles.output = hObject;
59
60 % Update handles structure
61 guidata(hObject, handles);
62
63 % UIWAIT makes trasteando wait for user response (see UIRESUME)
64 % uiwait(handles.figure1);
65
66
67 % --- Outputs from this function are returned to the command line.
68 function varargout = trasteando_OutputFcn(hObject, eventdata, handles)
69 % varargout  cell array for returning output args (see VARARGOUT);
70 % hObject    handle to figure
71 % eventdata  reserved - to be defined in a future version of MATLAB
72 % handles    structure with handles and user data (see GUIDATA)
73
74 % Get default command line output from handles structure
75 varargout{1} = handles.output;
76
77
78 % --- Executes on slider movement.
79 function sliderVientoEO_Callback(hObject, eventdata, handles)
80 % hObject    handle to sliderVientoEO (see GCBO)
81 % eventdata  reserved - to be defined in a future version of MATLAB
82 % handles    structure with handles and user data (see GUIDATA)
```

```
83
84  % Hints: get(hObject,'Value') returns position of slider
85  %        get(hObject,'Min') and get(hObject,'Max') to determine ...
        range of slider
86
87  %Máximo y mínimo del slider
88
89  set(hObject,'Max',5,'Min',-5);
90
91  %Redondeo del valor del slider y display como texto
92
93  valor=round(get(handles.sliderVientoEO,'Value'));
94
95  set(handles.numero_vientoEO,'String',valor);
96
97
98
99  % --- Executes during object creation, after setting all properties.
100 function sliderVientoEO_CreateFcn(hObject, eventdata, handles)
101 % hObject    handle to sliderVientoEO (see GCBO)
102 % eventdata  reserved - to be defined in a future version of MATLAB
103 % handles    empty - handles not created until after all CreateFcns ...
        called
104
105 % Hint: slider controls usually have a light gray background.
106 if isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
107     set(hObject,'BackgroundColor',[.9 .9 .9]);
108 end
109
110
111 % --- Executes on selection change in popupmenu2.
112 function popupmenu2_Callback(hObject, eventdata, handles)
113 % hObject    handle to popupmenu2 (see GCBO)
114 % eventdata  reserved - to be defined in a future version of MATLAB
115 % handles    structure with handles and user data (see GUIDATA)
116
117 % Hints: contents = cellstr(get(hObject,'String')) returns ...
        popupmenu2 contents as cell array
118 %        contents{get(hObject,'Value')} returns selected item from ...
        popupmenu2
119
120
121 % --- Executes during object creation, after setting all properties.
122 function popupmenu2_CreateFcn(hObject, eventdata, handles)
123 % hObject    handle to popupmenu2 (see GCBO)
124 % eventdata  reserved - to be defined in a future version of MATLAB
125 % handles    empty - handles not created until after all CreateFcns ...
        called
126
127 % Hint: popupmenu controls usually have a white background on Windows.
128 %       See ISPC and COMPUTER.
```

```
129 if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
130     set(hObject,'BackgroundColor','white');
131 end
132
133
134 % --- Executes on slider movement.
135 function sliderTiempo_Callback(hObject, eventdata, handles)
136 % hObject    handle to sliderTiempo (see GCBO)
137 % eventdata  reserved - to be defined in a future version of MATLAB
138 % handles    structure with handles and user data (see GUIDATA)
139
140 % Hints: get(hObject,'Value') returns position of slider
141 %        get(hObject,'Min') and get(hObject,'Max') to determine ...
        range of slider
142
143
144 %Se cogen los parámetros necesarios de la estructura handles
145
146 vientoX=round(get(handles.sliderVientoEO,'Value'));
147 vientoY=round(get(handles.sliderVientoNS,'Value'));
148
149 velocidad=handles.velocidad;
150
151 sX=str2double(get(handles.sX,'String'));
152 sY=str2double(get(handles.sY,'String'));
153 eX=str2double(get(handles.eX,'String'));
154 eY=str2double(get(handles.eY,'String'));
155
156 %Cálculo del tiempo total mediante la función calculateLimTiempo
157
158 limTiempo=calculateLimTiempo(velocidad, sX,sY,eX,eY, vientoX, vientoY);
159
160 %Redondeo y paso a minutos
161
162 limTiempo=round(limTiempo*60);
163
164 %Máximo y mínimo del slider
165
166 set(hObject,'Max',limTiempo,'Min',0);%Establecer dicho valor en el ...
        slider para esos datos
167
168 %Redondeo del valor del slider y display como texto
169
170 valor=round(get(handles.sliderTiempo,'Value'));
171 set(handles.numero_tiempo,'String',valor);
172
173
174
175
176
177
178
```

```
179 % --- Executes during object creation, after setting all properties.
180 function sliderTiempo_CreateFcn(hObject, eventdata, handles)
181 % hObject    handle to sliderTiempo (see GCBO)
182 % eventdata  reserved - to be defined in a future version of MATLAB
183 % handles    empty - handles not created until after all CreateFcns ...
       called
184
185 % Hint: slider controls usually have a light gray background.
186 if isequal(get(hObject,'BackgroundColor'), ...
       get(0,'defaultUicontrolBackgroundColor'))
187     set(hObject,'BackgroundColor',[.9 .9 .9]);
188 end
189
190
191 % --- Executes on selection change in popupmenu1.
192 function popupmenu1_Callback(hObject, eventdata, handles)
193 % hObject    handle to popupmenu1 (see GCBO)
194 % eventdata  reserved - to be defined in a future version of MATLAB
195 % handles    structure with handles and user data (see GUIDATA)
196
197 % Hints: contents = cellstr(get(hObject,'String')) returns ...
       popupmenu1 contents as cell array
198 %        contents{get(hObject,'Value')} returns selected item from ...
       popupmenu1
199
200
201 % --- Executes during object creation, after setting all properties.
202 function popupmenu1_CreateFcn(hObject, eventdata, handles)
203 % hObject    handle to popupmenu1 (see GCBO)
204 % eventdata  reserved - to be defined in a future version of MATLAB
205 % handles    empty - handles not created until after all CreateFcns ...
       called
206
207 % Hint: popupmenu controls usually have a white background on Windows.
208 %       See ISPC and COMPUTER.
209 if ispc && isequal(get(hObject,'BackgroundColor'), ...
       get(0,'defaultUicontrolBackgroundColor'))
210     set(hObject,'BackgroundColor','white');
211 end
212
213
214 % --- Executes on selection change in popupmenu3.
215 function popupmenu3_Callback(hObject, eventdata, handles)
216 % hObject    handle to popupmenu3 (see GCBO)
217 % eventdata  reserved - to be defined in a future version of MATLAB
218 % handles    structure with handles and user data (see GUIDATA)
219
220 % Hints: contents = cellstr(get(hObject,'String')) returns ...
       popupmenu3 contents as cell array
221 %        contents{get(hObject,'Value')} returns selected item from ...
       popupmenu3
222
223
```

```
224 % --- Executes during object creation, after setting all properties.
225 function popupmenu3_CreateFcn(hObject, eventdata, handles)
226 % hObject    handle to popupmenu3 (see GCBO)
227 % eventdata  reserved - to be defined in a future version of MATLAB
228 % handles    empty - handles not created until after all CreateFcns ...
       called
229
230 % Hint: popupmenu controls usually have a white background on Windows.
231 %       See ISPC and COMPUTER.
232 if ispc && isequal(get(hObject,'BackgroundColor'), ...
       get(0,'defaultUicontrolBackgroundColor'))
233     set(hObject,'BackgroundColor','white');
234 end
235
236
237 % --- Executes on slider movement.
238 function sliderVientoNS_Callback(hObject, eventdata, handles)
239 % hObject    handle to sliderVientoNS (see GCBO)
240 % eventdata  reserved - to be defined in a future version of MATLAB
241 % handles    structure with handles and user data (see GUIDATA)
242
243 % Hints: get(hObject,'Value') returns position of slider
244 %        get(hObject,'Min') and get(hObject,'Max') to determine ...
       range of slider
245
246 %Máximo y mínimo del slider
247
248 set(hObject,'Max',5,'Min',-5);
249
250 %Redondeo del valor del slider y display como texto
251
252 valor=round(get(handles.sliderVientoNS,'Value'));
253 set(handles.numero_vientoNS,'String',valor);
254
255
256
257 % --- Executes during object creation, after setting all properties.
258 function sliderVientoNS_CreateFcn(hObject, eventdata, handles)
259 % hObject    handle to sliderVientoNS (see GCBO)
260 % eventdata  reserved - to be defined in a future version of MATLAB
261 % handles    empty - handles not created until after all CreateFcns ...
       called
262
263 % Hint: slider controls usually have a light gray background.
264 if isequal(get(hObject,'BackgroundColor'), ...
       get(0,'defaultUicontrolBackgroundColor'))
265     set(hObject,'BackgroundColor',[.9 .9 .9]);
266 end
267
268
269
270 function sX_Callback(hObject, eventdata, handles)
271 % hObject    handle to sX (see GCBO)
```

```
272 % eventdata  reserved - to be defined in a future version of MATLAB
273 % handles    structure with handles and user data (see GUIDATA)
274
275 % Hints: get(hObject,'String') returns contents of sX as text
276 %        str2double(get(hObject,'String')) returns contents of sX as ...
        a double
277
278 matrix=handles.elevacion;
279
280 [dim1, dim2]=size(matrix);
281
282 sX=str2double(get(hObject,'String'));
283
284 %Se introducen restricciones de los valores que se pueden meter para que
285 %esten comprendidos entre 1 y dim1
286
287 if((sX>dim1)||(sX<1))
288 errordlg('Valor No Admitido','Curso_GUIDE');
289 end
290
291
292
293
294
295 % --- Executes during object creation, after setting all properties.
296 function sX_CreateFcn(hObject, eventdata, handles)
297 % hObject    handle to sX (see GCBO)
298 % eventdata  reserved - to be defined in a future version of MATLAB
299 % handles    empty - handles not created until after all CreateFcns ...
        called
300
301 % Hint: edit controls usually have a white background on Windows.
302 %       See ISPC and COMPUTER.
303 if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
304     set(hObject,'BackgroundColor','white');
305 end
306
307
308
309 function sY_Callback(hObject, eventdata, handles)
310 % hObject    handle to sY (see GCBO)
311 % eventdata  reserved - to be defined in a future version of MATLAB
312 % handles    structure with handles and user data (see GUIDATA)
313
314 % Hints: get(hObject,'String') returns contents of sY as text
315 %        str2double(get(hObject,'String')) returns contents of sY as ...
        a double
316
317 matrix=handles.elevacion;
318
319 [dim1, dim2]=size(matrix);
320
```

```
321 sY=str2double(get(hObject,'String'));
322
323 %Se introducen restricciones de los valores que se pueden meter para que
324 %esten comprendidos entre 1 y dim2
325 if((sY>dim2)||(sY<1))
326 errordlg('Valor No Admitido','Curso_GUIDE');
327 end
328
329 % --- Executes during object creation, after setting all properties.
330 function sY_CreateFcn(hObject, eventdata, handles)
331 % hObject    handle to sY (see GCBO)
332 % eventdata  reserved - to be defined in a future version of MATLAB
333 % handles    empty - handles not created until after all CreateFcns ...
       called
334
335 % Hint: edit controls usually have a white background on Windows.
336 %       See ISPC and COMPUTER.
337 if ispc && isequal(get(hObject,'BackgroundColor'), ...
       get(0,'defaultUicontrolBackgroundColor'))
338     set(hObject,'BackgroundColor','white');
339 end
340
341
342
343 function eX_Callback(hObject, eventdata, handles)
344 % hObject    handle to eX (see GCBO)
345 % eventdata  reserved - to be defined in a future version of MATLAB
346 % handles    structure with handles and user data (see GUIDATA)
347
348 % Hints: get(hObject,'String') returns contents of eX as text
349 %        str2double(get(hObject,'String')) returns contents of eX as ...
       a double
350
351 matrix=handles.elevacion;
352
353 [dim1, dim2]=size(matrix);
354
355 eX=str2double(get(hObject,'String'));
356 %Se introducen restricciones de los valores que se pueden meter para que
357 %esten comprendidos entre 1 y dim1
358 if((eX>dim1)||(eX<1))
359 %handles.inix=100;
360 errordlg('Valor No Admitido','Curso_GUIDE');
361 end
362
363 % --- Executes during object creation, after setting all properties.
364 function eX_CreateFcn(hObject, eventdata, handles)
365 % hObject    handle to eX (see GCBO)
366 % eventdata  reserved - to be defined in a future version of MATLAB
367 % handles    empty - handles not created until after all CreateFcns ...
       called
368
369 % Hint: edit controls usually have a white background on Windows.
```

```
370 %       See ISPC and COMPUTER.
371 if ispc && isequal(get(hObject,'BackgroundColor'), ...
       get(0,'defaultUicontrolBackgroundColor'))
372     set(hObject,'BackgroundColor','white');
373 end
374
375
376
377 function eY_Callback(hObject, eventdata, handles)
378 % hObject    handle to eY (see GCBO)
379 % eventdata  reserved - to be defined in a future version of MATLAB
380 % handles    structure with handles and user data (see GUIDATA)
381
382 % Hints: get(hObject,'String') returns contents of eY as text
383 %        str2double(get(hObject,'String')) returns contents of eY as ...
       a double
384
385 matrix=handles.elevacion;
386
387 [dim1, dim2]=size(matrix);
388
389 eY=str2double(get(hObject,'String'));
390
391 %Se introducen restricciones de los valores que se pueden meter para que
392 %esten comprendidos entre 1 y dim2
393
394 if((eY>dim2)||(eY<1))
395 errordlg('Valor No Admitido','Curso_GUIDE');
396 end
397
398 % --- Executes during object creation, after setting all properties.
399 function eY_CreateFcn(hObject, eventdata, handles)
400 % hObject    handle to eY (see GCBO)
401 % eventdata  reserved - to be defined in a future version of MATLAB
402 % handles    empty - handles not created until after all CreateFcns ...
       called
403
404 % Hint: edit controls usually have a white background on Windows.
405 %       See ISPC and COMPUTER.
406 if ispc && isequal(get(hObject,'BackgroundColor'), ...
       get(0,'defaultUicontrolBackgroundColor'))
407     set(hObject,'BackgroundColor','white');
408 end
409
410
411 % --- Executes on button press in simular.
412 function simular_Callback(hObject, eventdata, handles)
413 % hObject    handle to simular (see GCBO)
414 % eventdata  reserved - to be defined in a future version of MATLAB
415 % handles    structure with handles and user data (see GUIDATA)
416
417
418 %Se cogen los parámetros necesarios de la estructura handles
```

```
419
420 elevacion=handles.elevacion;
421 velocidad=handles.velocidad;
422 tiempo=round(get(handles.sliderTiempo,'Value'));
423 vientoX=round(get(handles.sliderVientoEO,'Value'));
424 vientoY=round(get(handles.sliderVientoNS,'Value'));
425 limTiempo=get(handles.sliderTiempo,'Max');
426 sX=str2double(get(handles.sX,'String'));
427 sY=str2double(get(handles.sY,'String'));
428 eX=str2double(get(handles.eX,'String'));
429 eY=str2double(get(handles.eY,'String'));
430
431 %Se preparan los ejes del GUI
432 axes(handles.axes1);
433
434 %Uso de la función ejecuta_FM
435 ejecuta_FM_prueba(elevacion, velocidad, tiempo, vientoX, vientoY, ...
       sX, sY, eX, eY, limTiempo)
436
437
438 % --- Executes on button press in buscarMapa.
439 function buscarMapa_Callback(hObject, eventdata, handles)
440 % hObject    handle to buscarMapa (see GCBO)
441 % eventdata  reserved - to be defined in a future version of MATLAB
442 % handles    structure with handles and user data (see GUIDATA)
443
444 %el archivo .mat que se coja ha de contener 2 matrices de las mismas
445 %dimensiones llamadas elevacion y velocidad
446
447 filename=uigetfile('*.mat','Elegir archivo .mat');%Menú para ...
       conseguir las matrices
448 S = load(filename);%Estructura que las carga en el workspace actual
449
450 %Se pasa a la estructura handles para que pueda ser utilizada en otras
451 %partes del GUI
452 handles.elevacion=S.elevacion;
453 handles.velocidad=S.velocidad;
454 guidata(hObject,handles);
455
456
457 % --- Executes on button press in sinPropagacion.
458 function sinPropagacion_Callback(hObject, eventdata, handles)
459 % hObject    handle to sinPropagacion (see GCBO)
460 % eventdata  reserved - to be defined in a future version of MATLAB
461 % handles    structure with handles and user data (see GUIDATA)
462
463 %Se preparan los ejes del GUI
464 axes(handles.axes1);
465
466 hold off
467
468 %Representación de la matriz elevacion con el colormap de montaña
469
```

```
470 mesh(handles.elevacion);
471
472 zlimits = [min(handles.elevacion(:)) max(handles.elevacion(:))];
473 demcmap(zlimits);
474
475
476 % --- Executes on button press in combustion.
477 function pushbutton4_Callback(hObject, eventdata, handles)
478 % hObject    handle to combustion (see GCBO)
479 % eventdata  reserved - to be defined in a future version of MATLAB
480 % handles    structure with handles and user data (see GUIDATA)
481
482 figure
483
484 velocidad=flip_vertical(handles.velocidad);%Giro de la imagen para ...
       que coincida con la orientación de la matriz de elevacion
485
486 imagesc(velocidad);
```

Referencias

[1] (2017, Jun.) El economista. [Online]. Available: http://www.eleconomista.es/

[2] (2017, Jun.) Ministerio de agricultura pesca, alimentación y medio ambiente. [Online]. Available: http://www.mapama.gob.es/es/

[3] A. Bargueno, "Modelización de un incendio forestal mediante fast marching," Master's thesis, Universidad Carlos III Madrid, 2015.

[4] R. Sun, S. K. Krueger, M. A. Jenkins, M. A. Zulauf, and J. J. Charney, "The importance of fireatmosphere coupling and boundary-layer turbulence to wildfire spread," *International Journal of Wildland Fire 18(1) 50-60*, 2009.

[5] J. S. Gould, N. Cheney, L. McCaw, and S. Cheney, "Effects of head fire shape and size on forest fire rate of spread," *Conference Proceedings of 3rd International Wildland Fire (pp. 3-6)*, 2003.

[6] M. Guijarro, J. Madrigal, C. Díez, and J. S. Martín, "Caracterización de la propagación del fuego en matorral de carqueixa mediante quemas en túnel de viento," *Congresos-CARGA FINAL*, 2005.

[7] P. H. Kourtz and O'Regan, "A model a small forest fire to simulate burned and burning areas for use in a detection model. forest science," *Forest Science, Volume 17, Number 2, 1 June 1971, pp. 163-169(7)*, 1971.

[8] V. Mallet, D. Keyes, and F. Fendell, "Modeling wildland fire propagation with level set methods," *Computers and Mathematics with Applications 57 1089-1101*, 2009.

[9] J. Hilton, C. Miller, A. Sullivan, and C. Rucinski, "Effects of spatial and temporal variation in environmental conditions on simulation of wildfire spread," *Environmental Modelling & Software 67 118-127*, 2015.

[10] R. C. Rothermel, "A mathematical model for predicting fire spread in wildland fuels." 1972.

[11] D. Morvan, J. Dupuy, E. Rigolot, and J. Valette, "Firestar: A physically based model to study wildfire behaviour," *Forest Ecology and Management 234S S114*, 2006.

[12] (2017, Jun.) Firers. [Online]. Available: http://www.fire-rs.com/es/

[13] C. Tymstra, R. Bryce, B. Wotton, S. Taylor4, and O. Armitage, "Development and structure of prometheus: the canadian wildland fire growth simulation model," Canadian Forest Service, Tech. Rep., 2010.

[14] *Burn-CP3 Version 4.7 User's manual, 2017.*

[15] M. A. Finney, "Farsite: Fire area simulator-model development and evaluation." 2004.

[16] (2017, Jun.) Fireorg. [Online]. Available: http://www.fire.org/

[17] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations." *Journal of computational physics, 79(1), 12-49.*, 1988.

[18] J. V. Gómez, "Fast marching methods in path and motion planning: Improvements and high-level applications," Ph.D. dissertation, Universidad Carlos III Madrid, 2015.

[19] (2017, Jun.) Numerical tours. [Online]. Available: http://www.numerical-tours.com/

[20] S. Garrido, L. Moreno, F. Martín, and D. Álvarez, "Fast marching subjected to a vector fieldpath planning method for mars rovers," *Expert Systems With Applications 78 334346*, 2017.

[21] (2017, Jun.) Mathworks. [Online]. Available: https://es.mathworks.com/

[22] (2017, Jun.) Land fire. [Online]. Available: https://www.landfire.gov/