581 Final Report

Caleb Cramer

December 10, 2025

1 Motivations and Problem Background

As a resident of the Pacific Northwest, I've watched the wildfire crisis evolve from a occasional occurrence into a annual disruption. For the last decade, the late summer has consistently brought weeks where the air quality index is so hazardous even basic outdoor activity is ill-advised. Beyond the public health impacts, this crisis is deeply personal: I have family and friends serving as wildland first responders, giving me a close perspective on the escalating ferocity and unpredictability of these events.

This phenomenon is entangled in politics, land rights, ecology and other domains but the physics is rooted in the complex, non-linear behavior of combustion and fluid dynamics. My motivation for this project is to attempt a quantitative understanding of these physics. The central challenge is to accurately model the coupled system of heat diffusion, wind-driven convection, and fuel depletion that dictates a fire's spread. I aim to leverage the advanced numerical techniques learned this quarter to discretize this challenging, coupled Reaction-Diffusion-Convection (RDC) system.

While real-world fire models are vast in complexity, this project focuses on the core mathematical physics governing fire spread based on topography, wind, and fuel. This approach required certain simplifications, similar to those outlined in the foundational works of the domain, such as the modeling decisions made by [2], which served as a major inspiration and technical reference for this implementation.

Goal: Model a wildfire (temperature and remaining fuel) which starts at a particular location $[x_{start}, y_{start}]$ across a span of time $[0, t_{end}]$.

2 Definitions and Explanations

Parameters

- $[x, y]$ = spatial domain in question
- $[0, t_{end}]$ = time evaluation range
- k = diffusion coefficient
- ϵ = the inverse of the activation energy of the fuel
- α = natural convection coefficient
- q = reaction heat
- u_{pc} = phase change threshold

Functions

- $u(x, y, t) = u_{x,y,t} = k\Delta u - \upsilon \cdot \nabla u + f(u, \beta) \rightarrow$ temperature at time t and location x,y
  - $f(u, \beta) = \mathcal{H}_{pc}(u)\beta e^{\frac{u}{1+\epsilon u}} - \alpha u \rightarrow$ nonlinear heat source
  - $\mathcal{H}_{pc}$ is the Heaviside function centered at u_{pc}
- $\beta(x, y, t) = \beta_{x,y,t} \rightarrow$ amount of fuel left at time t and location x,y
  - $g(u, \beta) = -\mathcal{H}_{pc}(u)\frac{\epsilon}{q}\beta e^{\frac{u}{1+\epsilon u}}$
- $\upsilon(x, y, t) = w(t) + \nabla Z(x, y) \rightarrow$ wind and topography vector field
  - $w(t) \rightarrow$ vector field correlating to the wind at time t
  - $Z(x, y) \rightarrow$ vector field correponding to topography at time t and location x,y

3 Mathematical Model: Coupled Reaction-Diffusion-Convection System

The wildfire spread is modeled by a system of two coupled Partial Differential Equations (PDEs) for temperature, $u_{x,y,t}$, and fuel concentration, $\beta_{x,y,t}$. This model captures the key physical phenomena: heat diffusion, advection (convection) due to wind/topography, and a nonlinear reaction source/sink term.

The system is given by:

$$\frac{\partial u}{\partial t} = k\Delta u - \upsilon \cdot \nabla u + f(u, \beta) \tag{1}$$

$$\frac{\partial \beta}{\partial t} = g(u, \beta) \tag{2}$$

where $k\Delta u$ is the diffusion term, $-\upsilon \cdot \nabla u$ is the convection/advection term, and $f(u, \beta)$ and $g(u, \beta)$ are the nonlinear reaction terms defined in the Problem Setup.

3.1 Nonlinear Reaction Approximation

The model uses the Heaviside step function, $\mathcal{H}_{pc}(u)$, which introduces a discontinuity at the phase change threshold u_{pc} (ignition temperature). To ensure the system is stable with standard ODE/PDE solvers (like the Runge-Kutta method used) without stalling at the discontinuity, the Heaviside function is replaced by a smoothed Heaviside function based on the sigmoid (hyperbolic tangent) function:

$$\mathcal{H}_{pc}(u) \approx H(u) = 0.5\,(\tanh(c(u - u_{pc})) + 1) \tag{3}$$

where $c = 50$ is a steepness parameter used in the implementation to approximate a sharp transition.

This was based on research after I ran into stiffness with my system which was causing LSODA and BDF to fail. I discovered the 'sigmoid' approach taken by Miao Cui et al. in [1].

4 Numerical Method: Method of Lines

The coupled PDE system is solved using the Method of Lines (MoL). This technique transforms the system of PDEs into a large system of ordinary differential equations (ODEs) by discretizing the spatial derivatives while leaving the time derivative continuous. I chose to use Finite Difference instead of Spectral Methods due to the non-linearity of $f(u, \beta)$ and $g(u, \beta)$ as well as the Neumann Boundary Conditions. We solved this exact situation in class but for this implementation I elected to start simple and iterate when possible. It also saved me the time and effort of converting back and forth from the spectral doamin to the spatial domain to account for the non-linear terms.

4.1 Spatial Discretization (Finite Differences)

The spatial domain is a $2L \times 2L$ square discretized on an $m \times m$ grid, resulting in a total of $N = m^2$ interior grid points. With $m = 250$, the system size for u and β is $2 \times N = 125,000$. The spatial step size is $\delta = \frac{2L}{m-1}$.

4.1.1 Diffusion Operator (Laplacian)

The 2D Laplacian operator, $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$, is constructed using the Kronecker product of sparse 1D finite difference matrices.

- 1D Laplacian (D_{2_1d}): A three-point stencil (similar to HW3) is used for the 1D second derivative.
- Boundary Conditions: Neumann Boundary Conditions ($\nabla u \cdot \mathbf{n} = 0$, or zero flux at the edges) are enforced by modifying the first and last rows of the 1D operator, effectively mirroring the internal points. I had to add these manually when I noticed Setup 2 (described later) was causing issues at the boundaries. I used a flattened mask to set all boundary points to 0.
- 2D Laplacian (L_{2D}): The final operator is $L_{2D} = (I \otimes D_{2\_1d}) + (D_{2\_1d} \otimes I)$, where I is the $m \times m$ identity matrix and $\otimes$ is the Kronecker product. This method correctly applies the derivatives across the flattened m^2 vector of grid points.

4.1.2 Convection Operators (Gradient)

The 2D gradient operators ($\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y})$) are constructed using the Central Difference scheme for first derivatives.

- 1D Gradient (D_{1_1d}): Uses the stencil $\frac{1}{2\delta}(u_{i+1} - u_{i-1})$.
- 2D Gradient (D_x, D_y): $D_x = I \otimes D_{1\_1d}$ and $D_y = D_{1\_1d} \otimes I$.

The advection term is computed by element-wise multiplication with the velocity components:

$$\mathbf{v} \cdot \nabla u \approx V_x \odot (\mathbf{D_x} u) + V_y \odot (\mathbf{D_y} u) \tag{4}$$

where V_x and V_y are the flattened m^2 vectors of the velocity field $v(x, y, t)$ components.

4.2 Time Integration (ODE Solver)

The resulting system of ODEs, $\frac{d\mathbf{Y}}{dt} = F(t, \mathbf{Y})$, where $\mathbf{Y} = [\mathbf{u}; \beta]$, is solved using `scipy.integrate.solve_ivp` with the RK23 (Runge-Kutta of order 2(3)) method. I tested with RK45 and didn't get any different results.

5 Implementation and Results

5.1 Key Parameters

The following dimensionless parameters were used in the simulation:

- Grid Size (m): 250
- Domain Length (L): 20.0 (Domain is $[-20, 20] \times [-20, 20]$)
- Diffusion Coefficient (k): 0.1
- Inverse Activation Energy (ϵ): 0.2
- Natural Convection Coefficient (α): 0.05
- Reaction Heat (q): 1.0
- Phase Change Threshold (u_{pc}): 0.1
- Time Span (t_{span}): 8

5.2 Initial Conditions

- Initial Spark at (-5,5) (setup 1): $u_{x,y,0} = 3e^{-\frac{(X+5)^2+(Y+5)^2}{2}}$
- Initial Spark at (-15,0) (setup 2): $u_{x,y,0} = 3e^{-\frac{(X+15)^2+Y^2}{2}}$
- Fuel Distribution: $\beta_{x,y,0} = 1$
- Constant ENE wind: $x_{mag} = 0.8$ and $y_{mag} = 0.3$

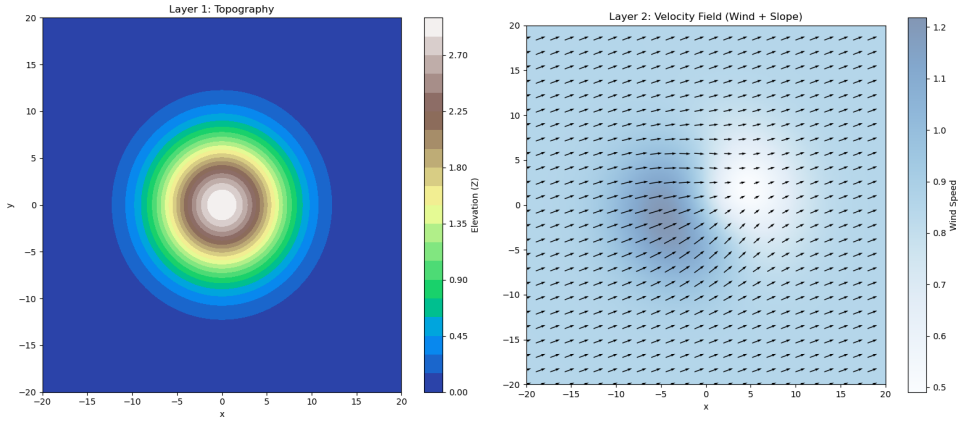

Figure 1: Initial setup for a hill of height 3 and radius 5 as defined by $Z_{hill} = 3 - 3e^{-\frac{X^2+Y^2}{50}}$

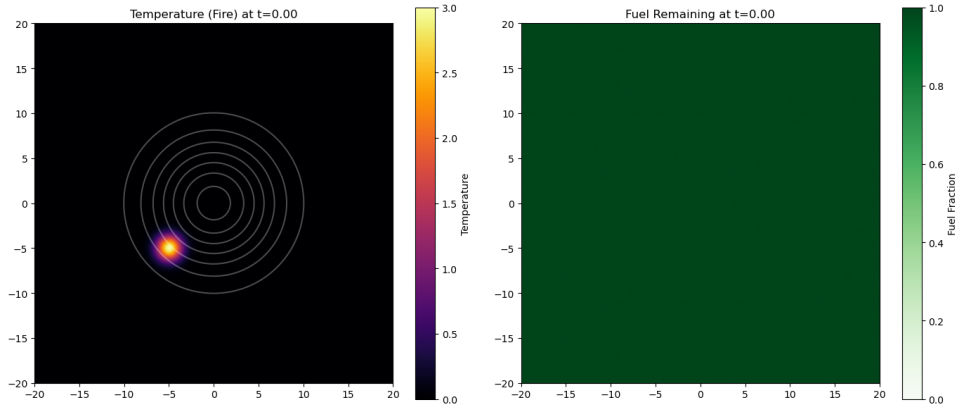

Figure 2: Initial state of the fire with setup 1. Left shows an initial spark and the evolution of $u(x, y, t)$ and right shows $\beta(x, y, t)$, the fuel remaining.

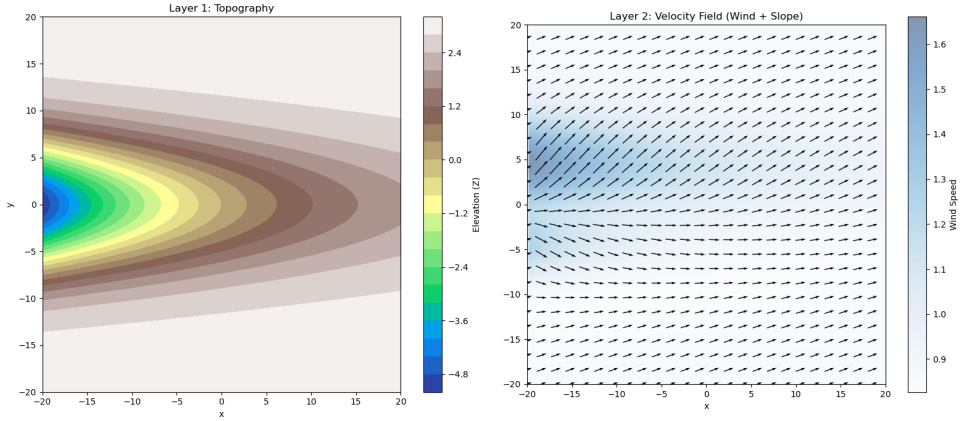

Figure 3: Initial setup for a valley of depth 5 as defined by $Z_{valley} = 3 - 3e^{-\frac{\frac{1}{2}X+\frac{1}{5}Y^2}{10}}$

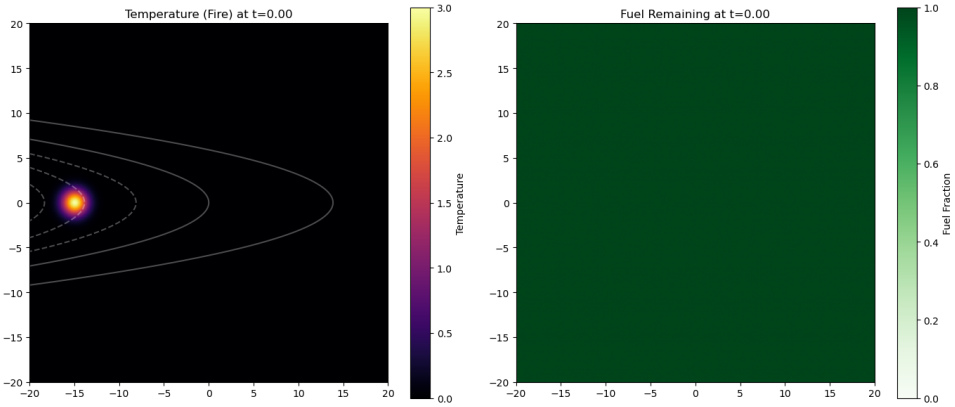

Figure 4: Initial state of the fire with setup 2, a valley mirrored about the y-axis and a new spark location. Left shows an initial spark and the evolution of $u(x, y, t)$ and right shows $\beta(x, y, t)$, the fuel remaining.

5.3 Numerical Performance

The problem was discretized into a system of $2 \times 250^2 = 125,000$ ODEs. The full system size for the linear operators (like the Laplacian) is $(62500, 62500)$. The sparsity of the Finite Difference operators allows for efficient matrix-vector products. The solution was obtained in 3.26 seconds using the `solve_ivp` routine.

5.4 Time Series Plots

See Figure 5 for initial state of the system under setup 1, Figure 6 for $t = 3.79$, and Figure 7 for $t = 8.00$. See Figure 8 for initial state of the system under setup 2, Figure 9 for $t = 3.79$, and Figure 10 for $t = 8.00$.

6 Core Implementation Code

The central part of the Method of Lines implementation is the Right Hand Side (RHS) function, which computes the time derivatives $\frac{du}{dt}$ and $\frac{d\beta}{dt}$ for the ODE solver.

```
def smoothed_heaviside(u, threshold, c=50):
    # Sigmoid approximation of Heaviside step function.
    return 0.5 * (np.tanh(c * (u - threshold)) + 1)

def reaction_rate(u, beta):
    # Arrhenius term: exp(u / (1 + eps*u))
    arrhenius = np.exp(u / (1 + epsilon * u))
    # Threshold: Fire only burns if u > u_pc
    H = smoothed_heaviside(u, u_pc)
    return H * beta * arrhenius

def model_rhs(t, state):
    """
    Right Hand Side of the PDE system.
    State is a vector of size 2*m*m (Temperature followed by Fuel).
    """
    n = m * m
    u = state[:n]      # Temperature (vectorized)
    beta = state[n:]   # Fuel (vectorized)

    # 1. Diffusion: k * Laplacian * u
    diff_term = k * (Laplacian @ u)

    # 2. Convection: - v . grad(u)
    adv_term = - (Vx * (Dx @ u) + Vy * (Dy @ u))

    # 3. Reaction Term (rr = H * beta * exp(...) )
    rr = reaction_rate(u, beta)

```

```
    # Heat Source: f_u = Reaction - Cooling
    f_u = rr - alpha * u

    # Fuel Consumption: g_beta = -(epsilon / q) * rr
    g_beta = -(epsilon / q) * rr

    du_dt = diff_term + adv_term + f_u
    dbeta_dt = g_beta

    return np.concatenate([du_dt, dbeta_dt])
```

7 Discussion and Extensions

I heavily relied on concepts from this quarter to build this project, including:

- Python Basics and Visualization
  - I utilized Python's libraries to generate informative plots and GIFs, essential for visualizing the 3D time evolution of the temperature and the underlying vector fields that drive the fire spread.
  - Efficient matrix operations from NumPy and SciPy were crucial for rapidly calculating the derivatives across the entire computational grid.
  - The proper reshaping of multi-dimensional arrays and vectors was required to translate the $2D$ spatial grid data into the $1D$ vector format necessary for the ODE solver.
- Finite Difference Schemes
  - A meshgrid of points $[X, Y]$ was created to accurately discretize the spatial domain and evaluate the temperature and fuel functions across the entire landscape.
  - Sparse matrices were generated to efficiently approximate the spatial derivatives (Laplacian and Gradient) at every point $[X, Y]$ within the domain using the central difference method.
  - Updating boundary values for both the temperature field (Neumann, zero-flux) and the topographical gradient (zero-slope) was implemented to correspond with the physical application requirements of the model.
- System of ODEs with Continuous Time
  - The Method of Lines (MOL) was applied to convert the partial differential equations (PDEs) into a large linear system of ordinary differential equations (ODEs) that could be solved over time.
  - I selected an appropriate ODE solver from scipy, specifically the RK23 method, based on the system's size and the necessity to balance computational cost with solution stability.

Extensions

- Utilize Chebyshev Polynomials instead of Finite Differences
  - A significant numerical extension would be to replace the Finite Difference schemes with Chebyshev Polynomials, which are part of the Spectral Methods family, to potentially achieve faster convergence and much higher accuracy for smooth solutions.
- Implement Experiment-Based Complexity to Input Equations and Variables
  - I made a couple of overly large simplifications when modeling the ignition process and the energy required for the phase change of the fuel; to be truly accurate, I would need to add a third ODE to explicitly model the moisture content and its evaporation.
  - My initial conditions defined the surface fuel as equal to 1 everywhere, representing 'perfect' wildfire conditions; I would like to integrate real-world fuel models from specialized tools such as FastFuels or LANDFIRE to increase the model's applicability and customization for varied environments.

  - My topographical constructions, such as the Gaussian hill, are crude and overly simplistic; a substantial improvement would involve importing $3D$ raster files from the USGS (Digital Elevation Models or DEMs) to use real-world landscapes for modeling fire start and growth scenarios.

- Include Time, Stability, and Complexity Analysis
  - A rigorous academic extension would be to perform a formal Time, Stability, and Complexity Analysis of the numerical scheme, assessing factors like the Courant-Friedrichs-Lewy (CFL) condition and the stiffness of the system.

- Optimize for GPU using CuPy or TensorFlow
  - Assuming I make all of the other enhancements listed above, and the scenarios I need to solve expand in size, the resulting system of ODEs will become computationally expensive very quickly, making acceleration with one or more GPUs using libraries like CuPy or TensorFlow a necessity to significantly reduce the time required per scenario.

8 Figures

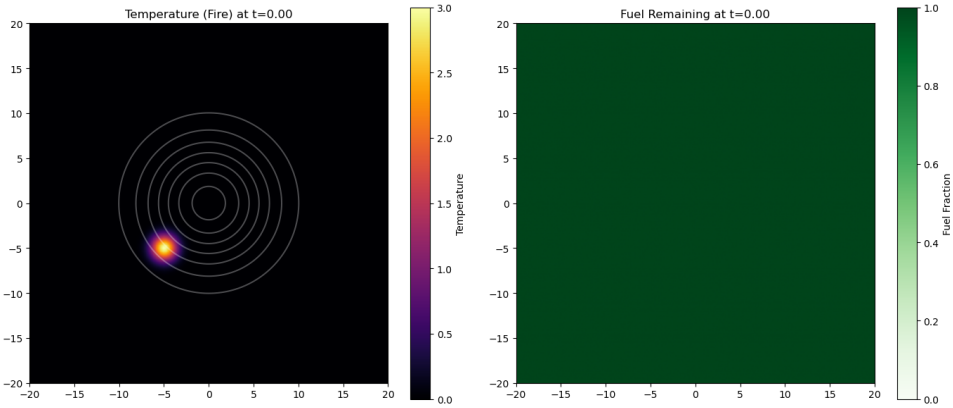

Figure 5: Initial state of the fire with setup 1. Left shows an initial spark and the evolution of $u(x, y, t)$ and right shows $\beta(x, y, t)$, the fuel remaining.

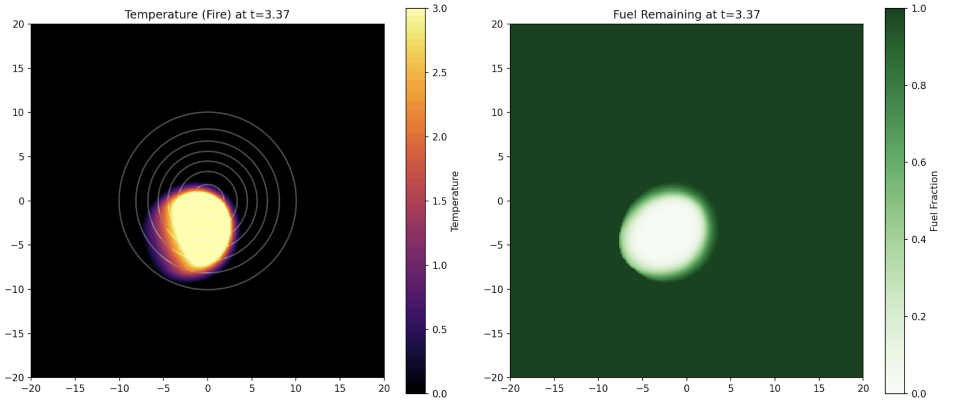

Figure 6: State of the fire in the middle of the simulation of setup 1. Left shows the evolution of $u(x, y, t)$ and right shows $\beta(x, y, t)$, the fuel remaining/consumed.

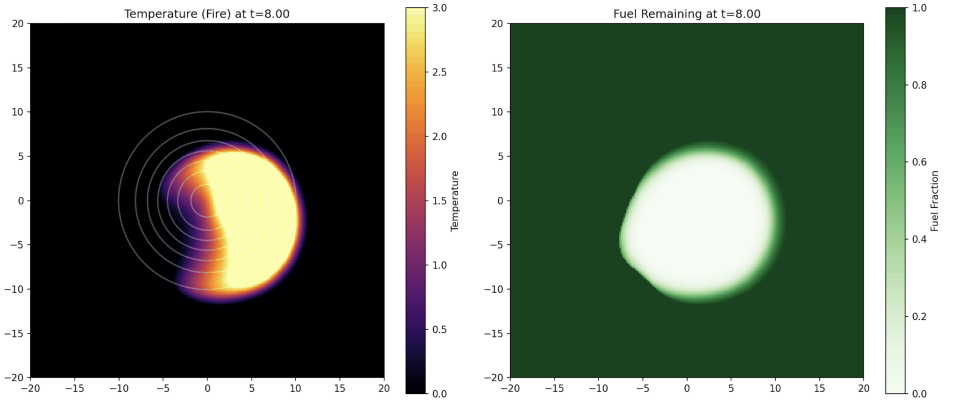

Figure 7: Final state of the fire with setup 1. Left shows the evolution of $u(x, y, t)$ and right shows $\beta(x, y, t)$, the fuel remaining/consumed.

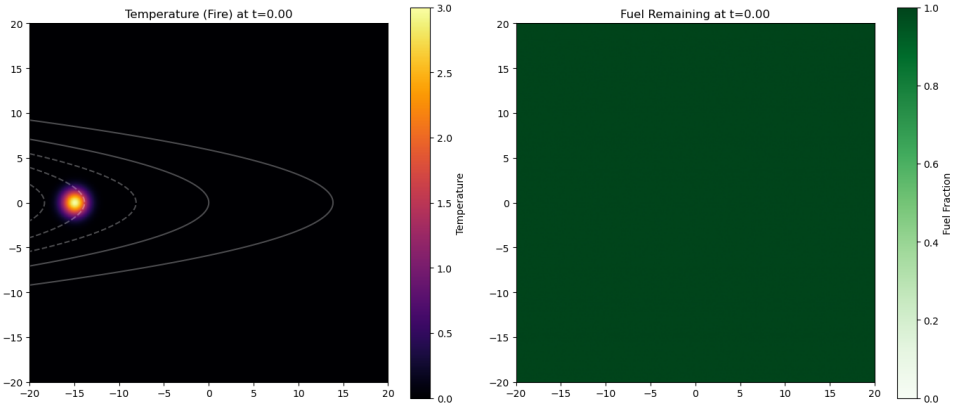

Figure 8: Initial state of the fire with setup 2, a valley mirrored about the y-axis and a new spark location. Left shows an initial spark and the evolution of $u(x, y, t)$ and right shows $\beta(x, y, t)$, the fuel remaining.

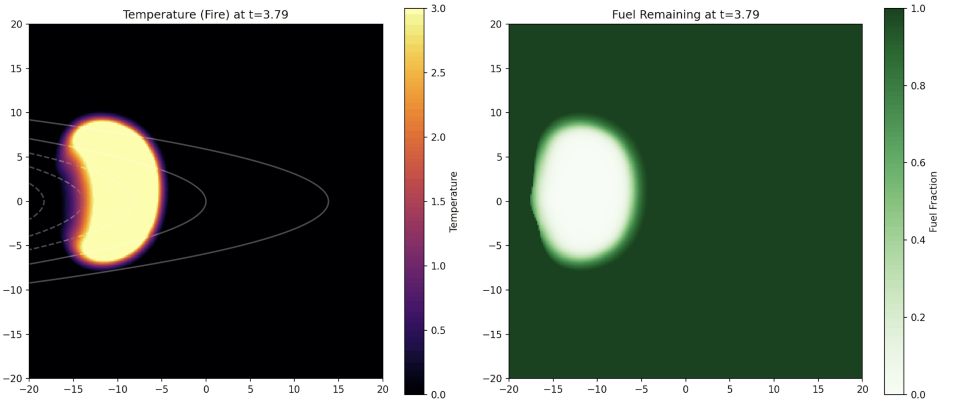

Figure 9: State of the fire in the middle of the simulation with setup 2. Left shows the evolution of $u(x, y, t)$ and right shows $\beta(x, y, t)$, the fuel remaining/consumed.

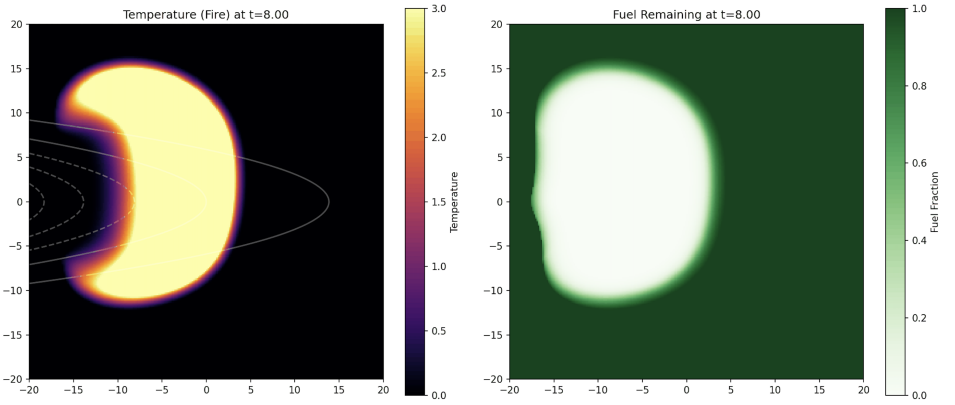

Figure 10: Final state of the fire with setup 2. Left shows the evolution of $u(x, y, t)$ and right shows $\beta(x, y, t)$, the fuel remaining/consumed.

References

[1] Miao Cui et al. "Numerical solution of phase change heat transfer problems by effective heat capacity model and element differential method". In: *Journal of Computational Science* 60 (2022), p. 101593. ISSN: 1877-7503. DOI: https://doi.org/10.1016/j.jocs.2022.101593. URL: https://www.sciencedirect.com/science/article/pii/S187775032200028X.

[2] Daniel San Martin and Claudio Torres. "2D Simplified Wildfire Spreading Model in Python: From NumPy to CuPy". In: *CLEI Electronic Journal* 26 (May 2023). DOI: 10.19153/cleiej.26.1.5.