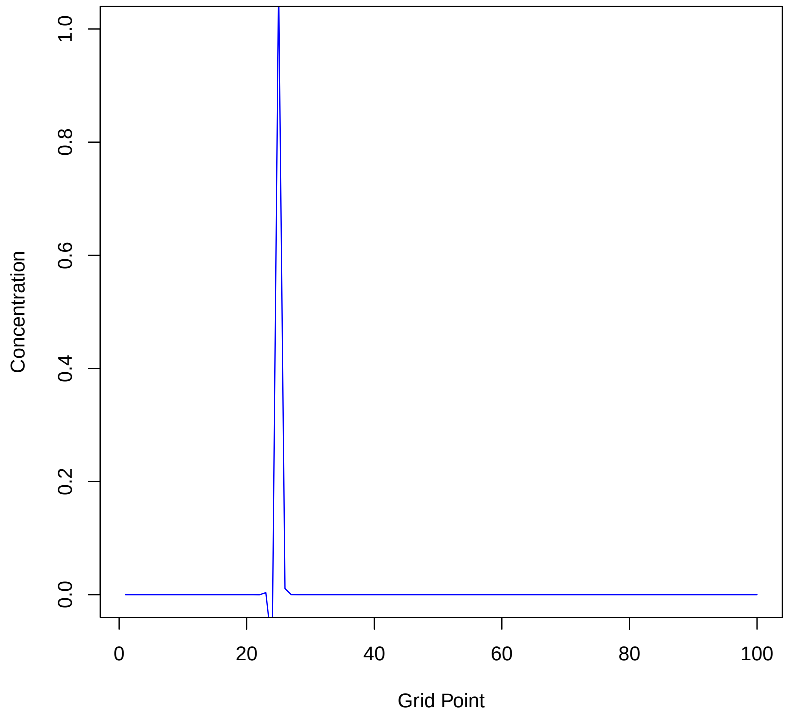
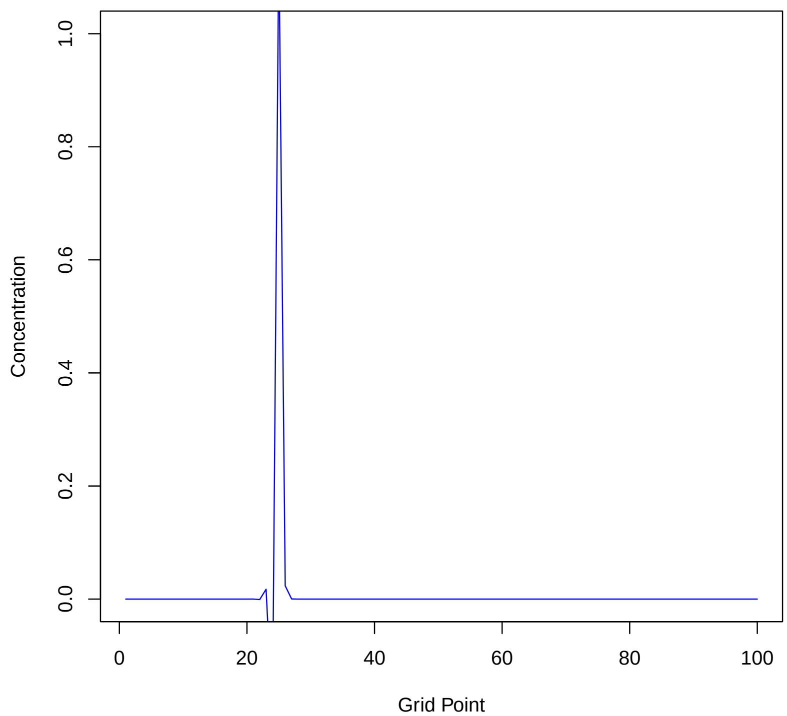
Advection and Diffusion

December 15, 2023

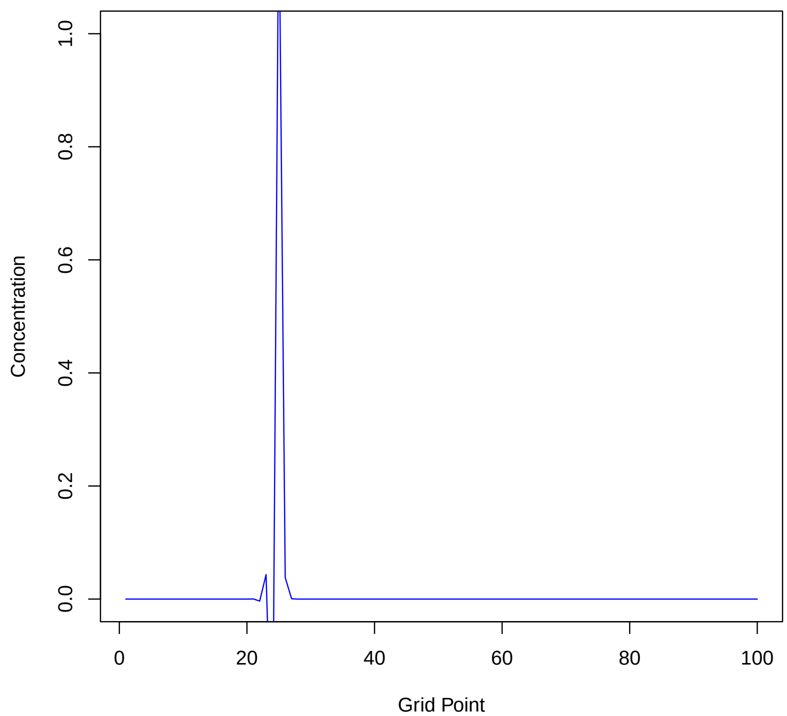
```r
# Advection and Diffusion Modeling in R

# Parameters
grid_size <- 100   # Number of grid points
time_steps <- 100  # Number of time steps
dt <- 0.1          # Time step size
dx <- 1            # Grid spacing
velocity <- 0.1    # Advection velocity
diffusion_coeff <- 0.01  # Diffusion coefficient

# Initialize grid
grid <- numeric(grid_size)
grid[ceiling(grid_size / 4)] <- 1  # Initial concentration at one-fourth of the grid

# Function to plot the current state of the grid
plot_grid <- function(grid) {
  plot(grid, type = 'l', col = 'blue', ylim = c(0, 1), xlab = 'Grid Point', ylab = 'Concentration')
}

# Main simulation loop
for (t in 1:time_steps) {
  # Advection
  advected <- c(grid[-1], grid[1])  # Shift concentration to the right (periodic boundary)
  grid <- grid - velocity * (advected - grid) * dt / dx

  # Diffusion
  diffused <- diffusion_coeff * (c(grid[-1], grid[1]) - 2 * grid + c(grid[grid_size], grid[-grid_size]))
  grid <- grid + diffused * dt / (dx^2)

  # Plot the current state of the grid every 10 time steps
  if (t %% 10 == 0) {
    plot_grid(grid)
    Sys.sleep(0.1)  # Pause to visualize the animation
  }
}
```

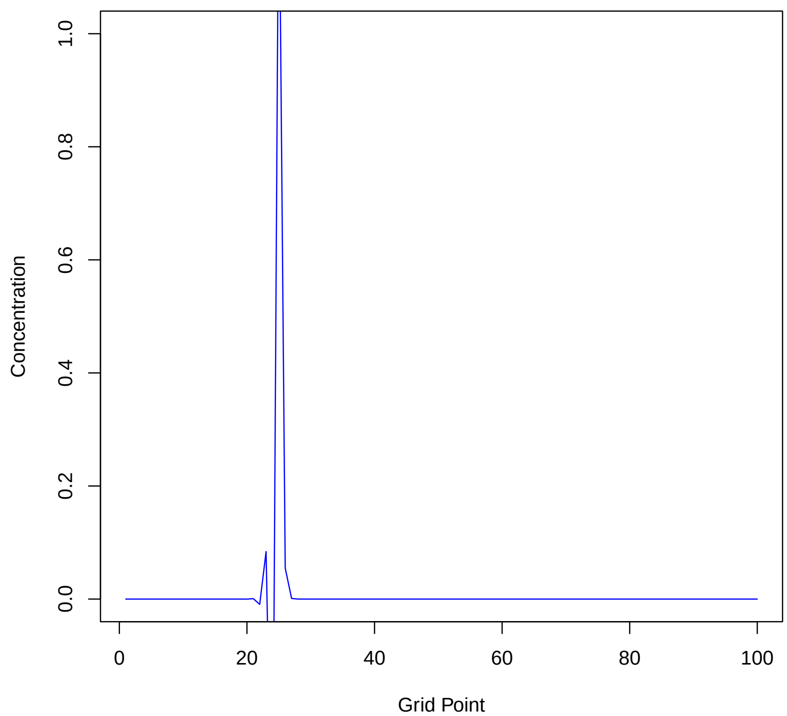


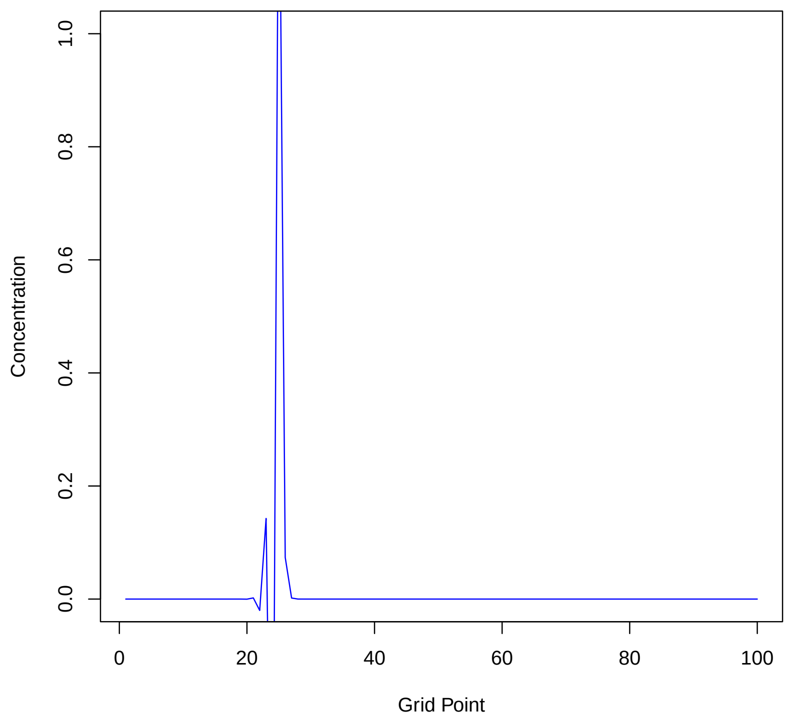
Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Grid Point



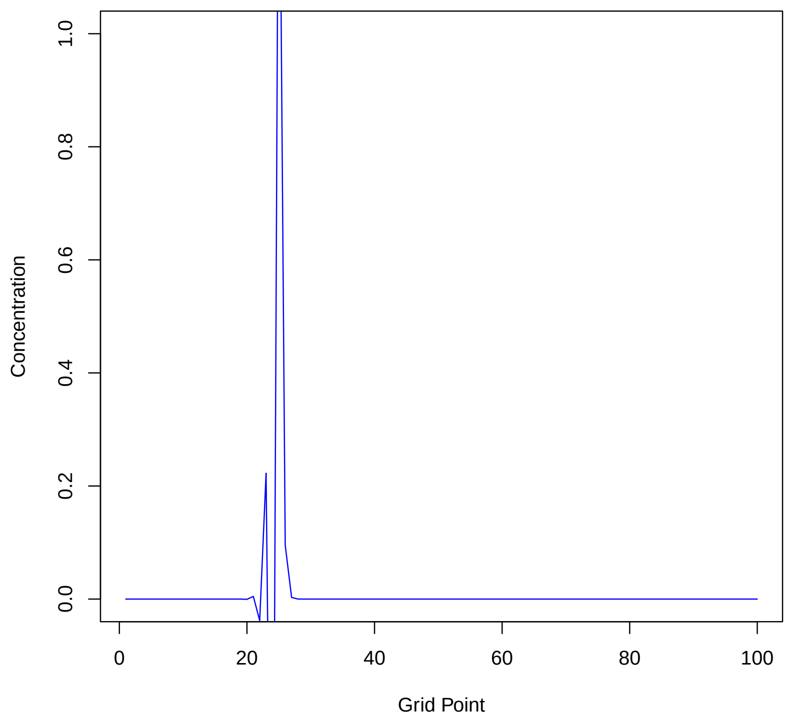
Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Grid Point


Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Grid Point

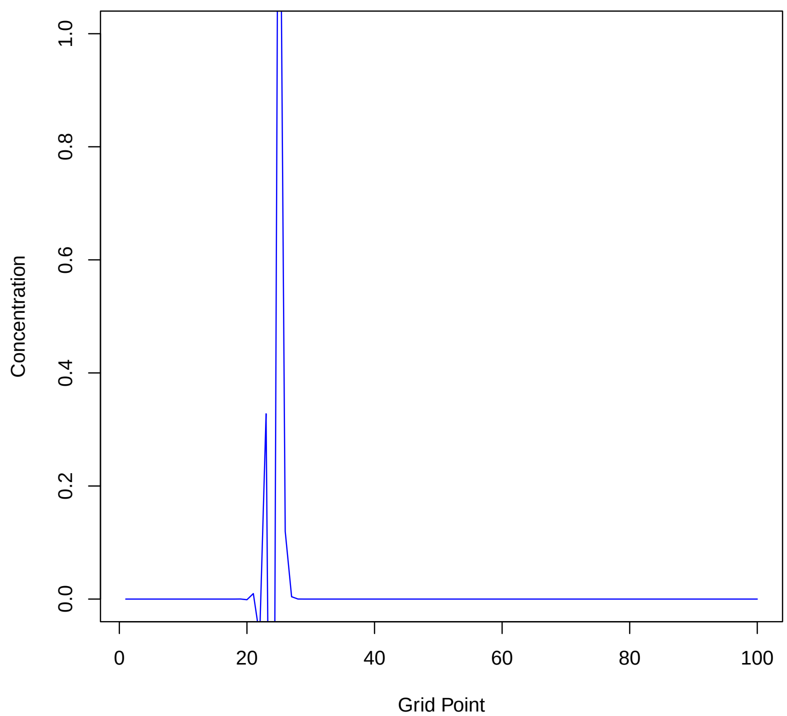

Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Grid Point

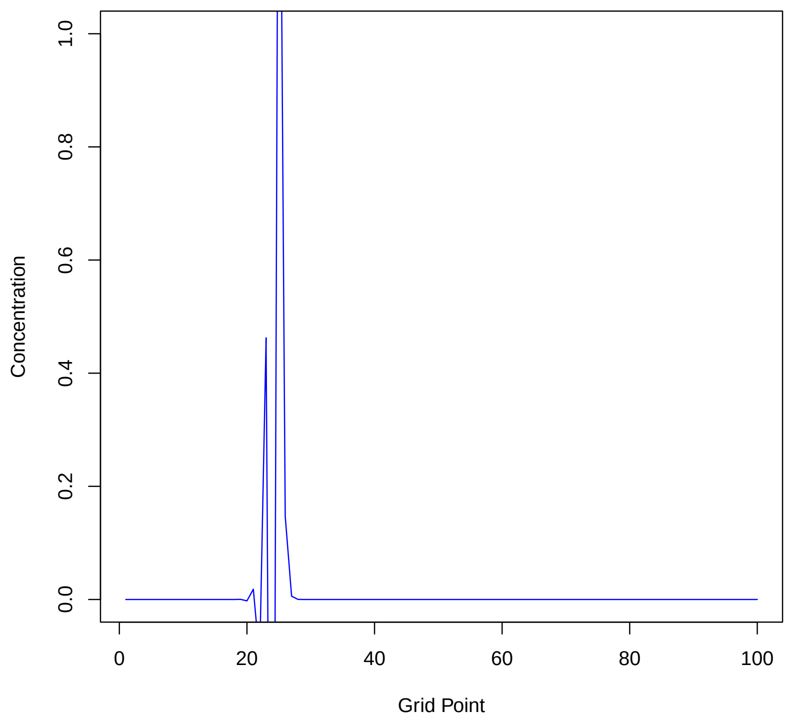


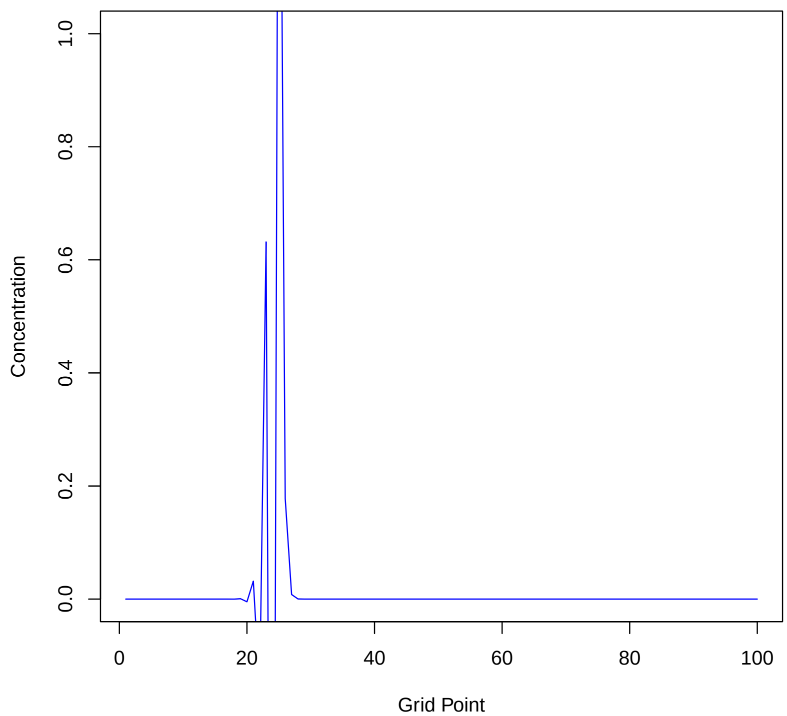
Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Grid Point



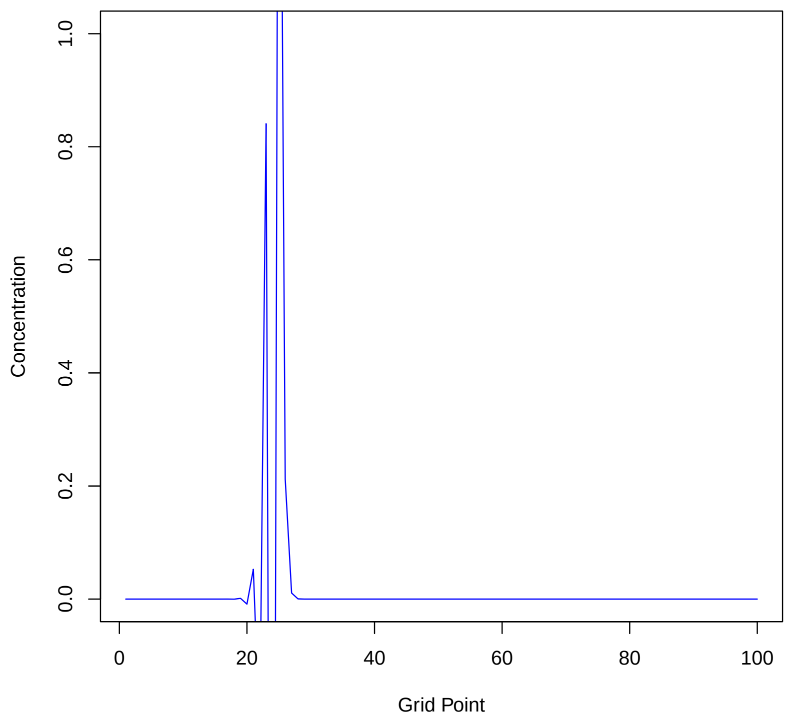
Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Grid Point


Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Grid Point


Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Grid Point



Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Grid Point

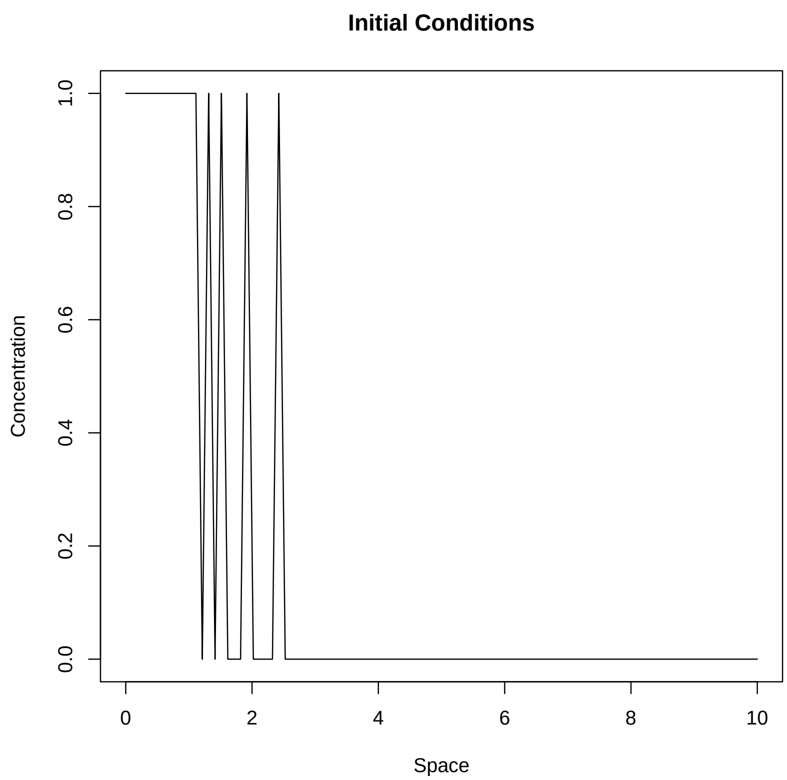



```r
# Final plot
plot_grid(grid)
```

Modeling Advection and Diffusion

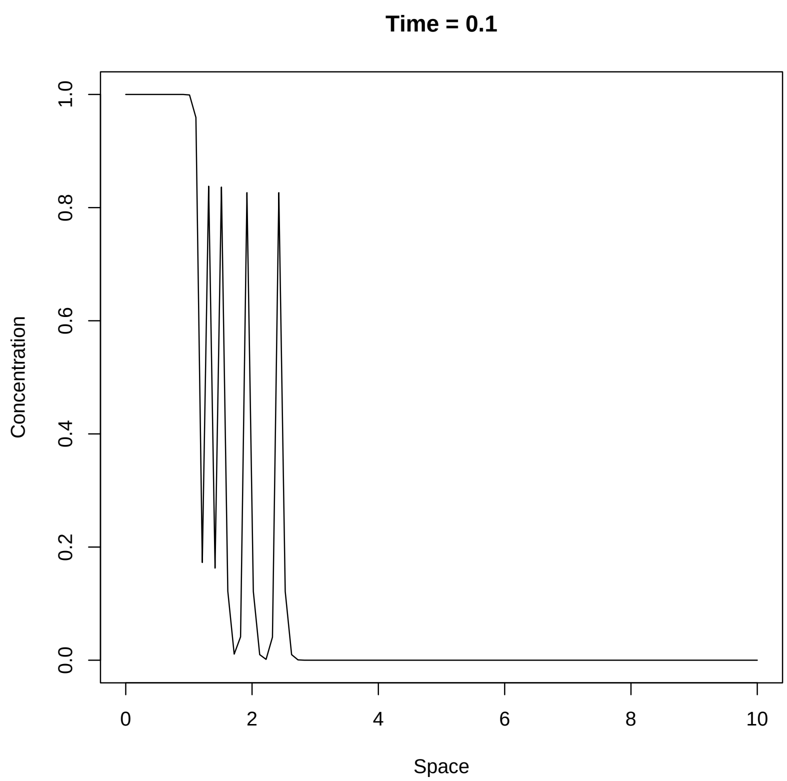
```r
# Advection-Diffusion Modeling in R

# Parameters
L <- 10          # Length of the domain
T <- 1           # Total simulation time
Nx <- 100         # Number of spatial grid points
Nt <- 100         # Number of time steps
alpha <- 0.01     # Diffusion coefficient
beta <- 0.1       # Advection coefficient

# Discretization
dx <- L / (Nx - 1)     # Spatial step size
dt <- T / Nt           # Time step size
```

```
# Initial conditions
u <- rep(0, Nx)
u[Nx %/% 4:(3 * Nx %/% 4)] <- 1  # Initial pulse in the middle of the domain

# Plot initial conditions
plot(seq(0, L, length.out = Nx), u, type = "l", ylim = c(0, 1), xlab = "Space", ylab = "Concentration", ma
```

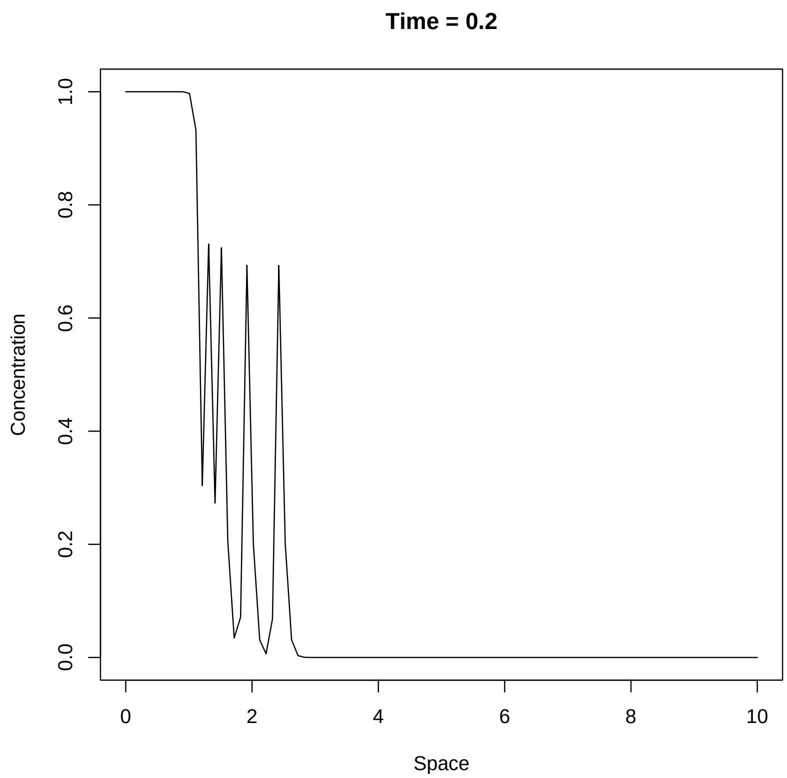


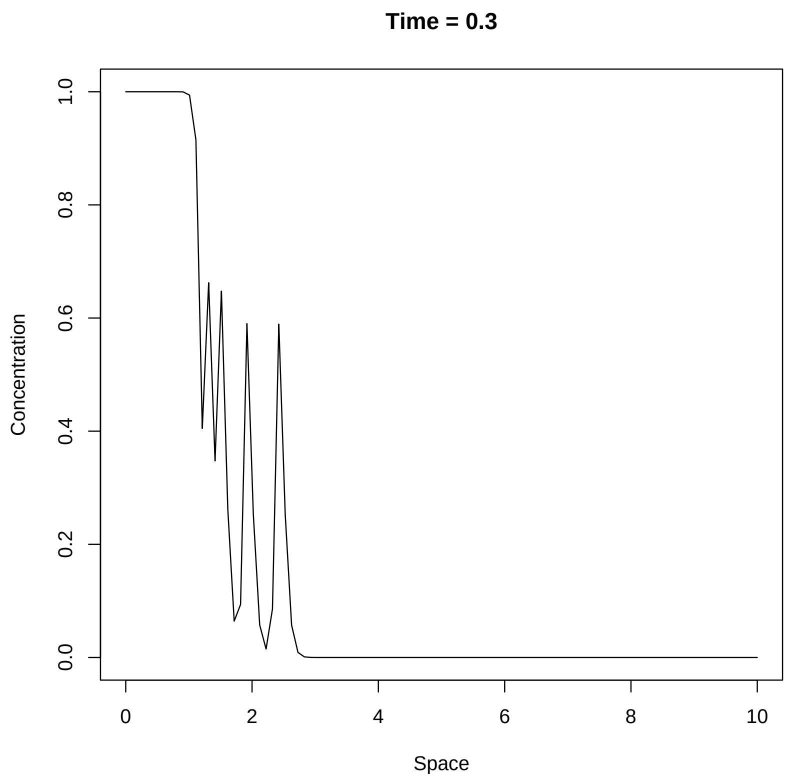

```
# Advection-Diffusion simulation using finite difference method
for (t in 1:Nt) {
  for (x in 2:(Nx - 1)) {
    u[x] <- u[x] + alpha * (u[x + 1] - 2 * u[x] + u[x - 1]) * dt / dx^2 - beta * (u[x + 1] - u[x - 1]) * d
  }

  # Apply boundary conditions (zero-flux)
  u[1] <- u[2]
  u[Nx] <- u[Nx - 1]

  # Plot the current state every 10 time steps
  if (t %% 10 == 0) {
    plot(seq(0, L, length.out = Nx), u, type = "l", ylim = c(0, 1), xlab = "Space", ylab = "Concentration"
    Sys.sleep(0.1)  # Pause to visualize the animation
```

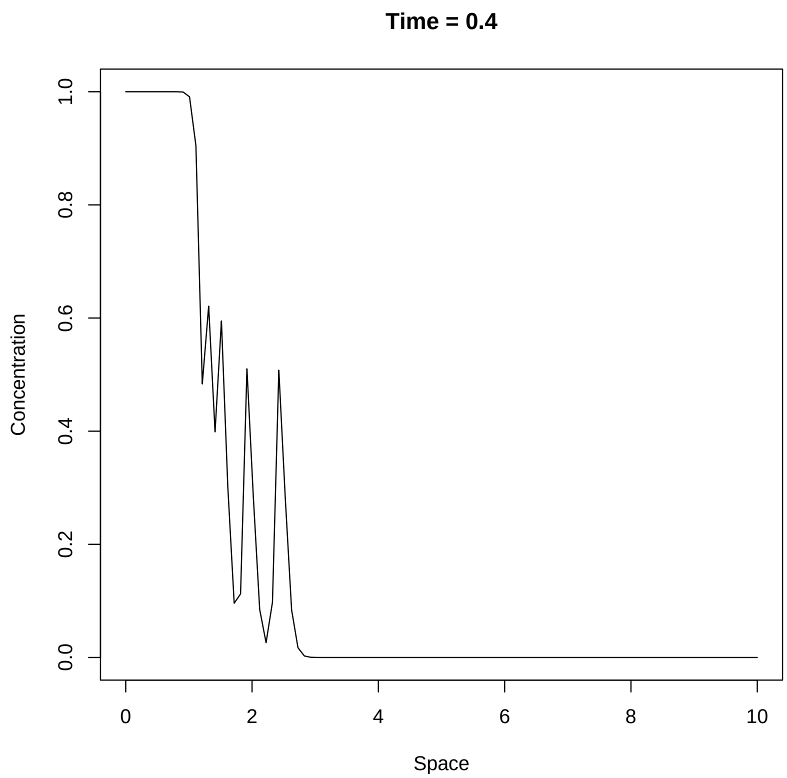
```
  }
}
```

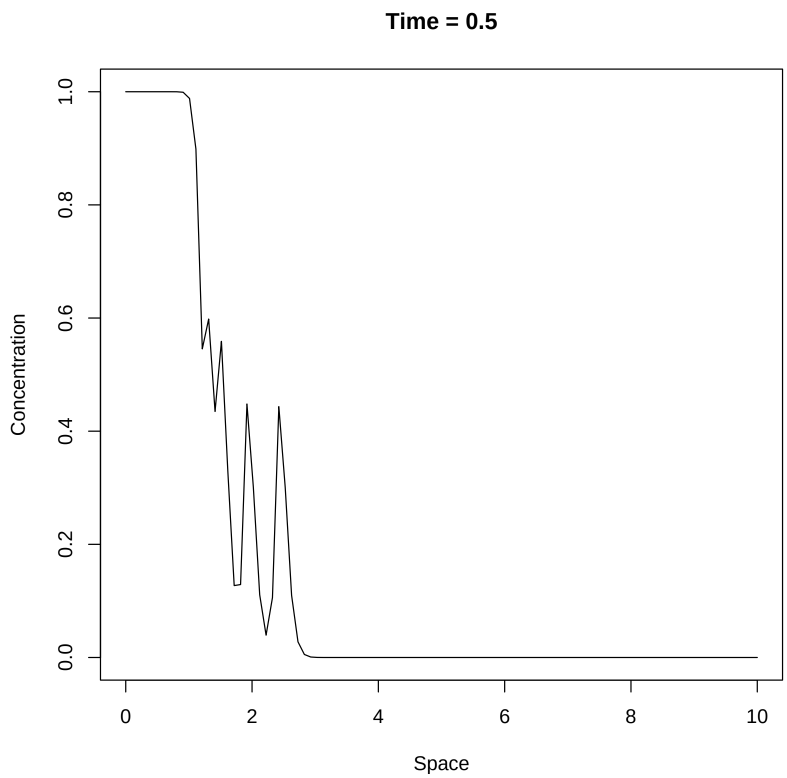


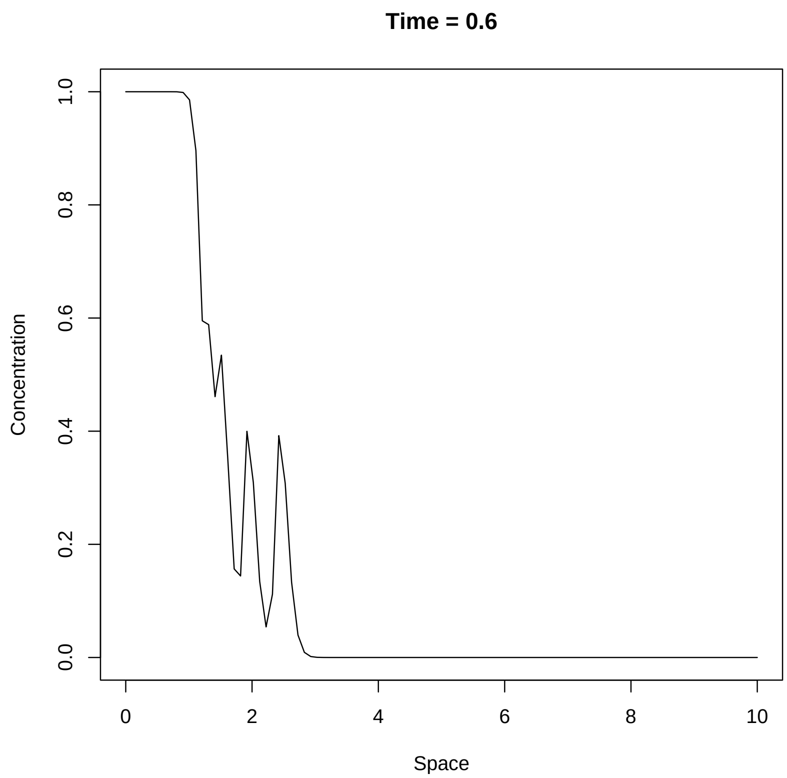
Time = 0.1
Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
Space

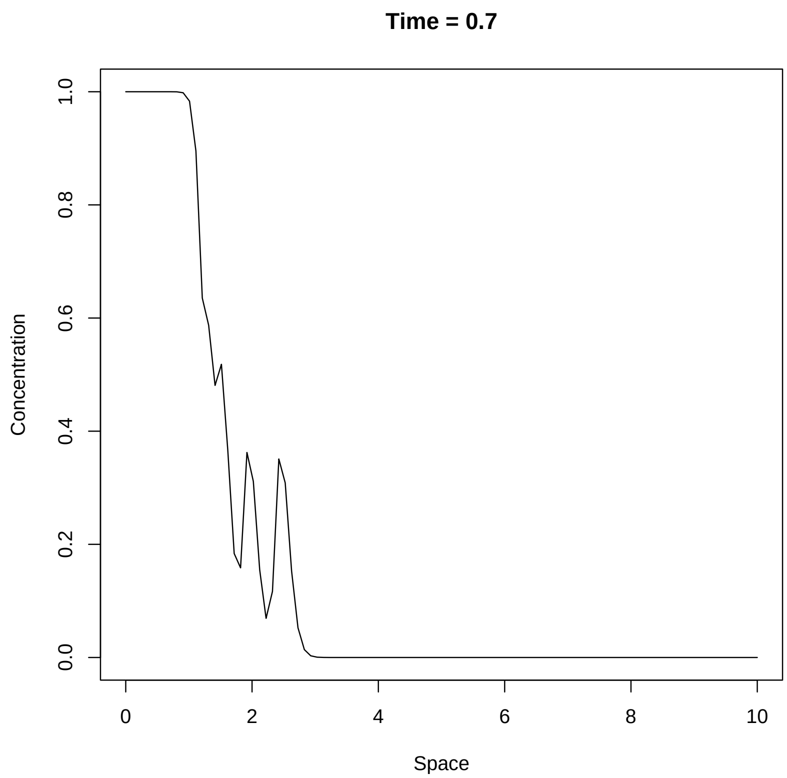

Time = 0.2
Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
Space

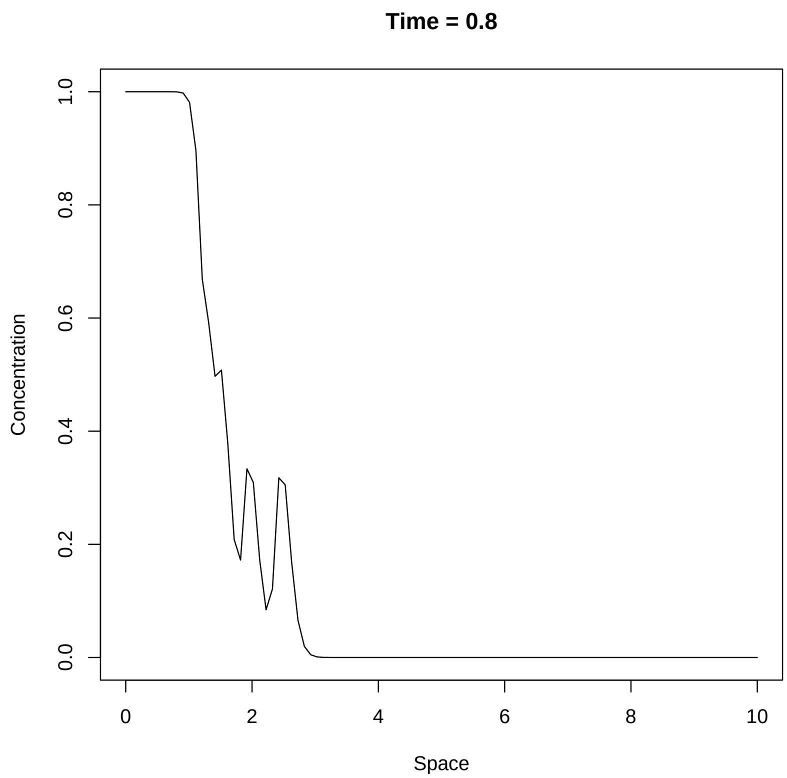

Time = 0.3
Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
Space

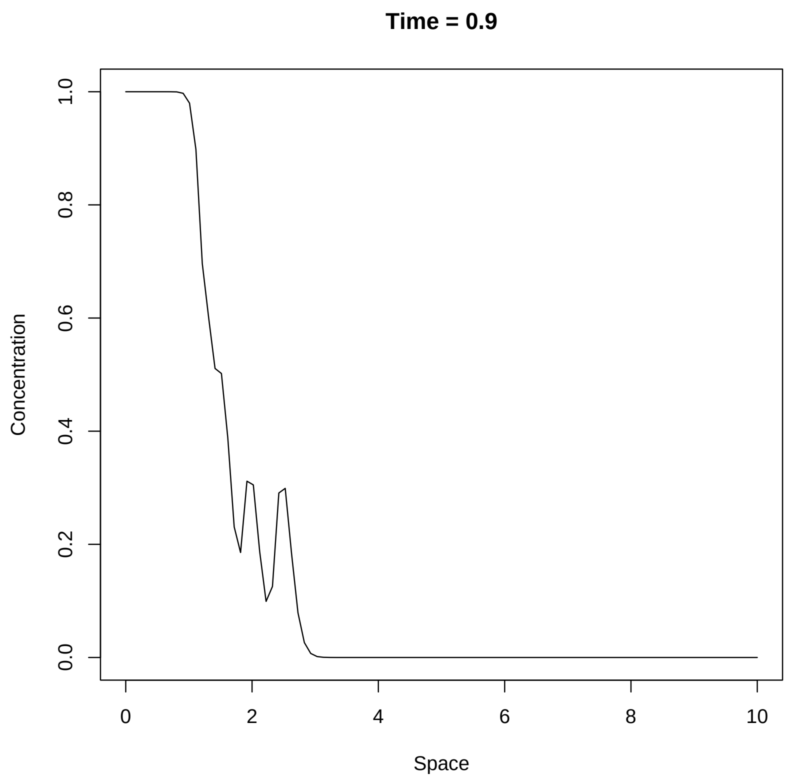

Time = 0.4
Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
Space

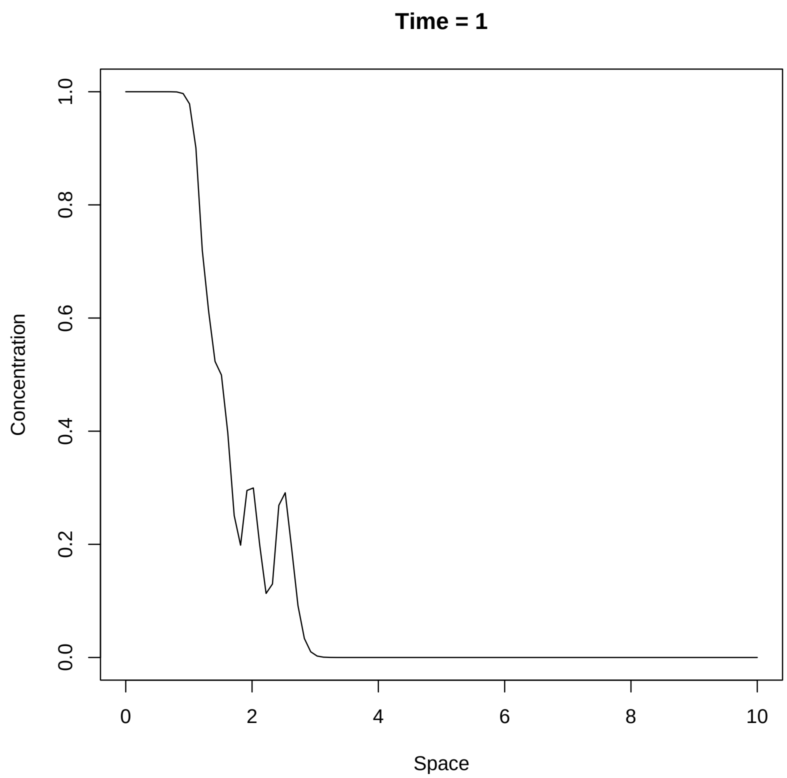

Time = 0.5
Concentration
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
Space


Time = 0.6
Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
Space

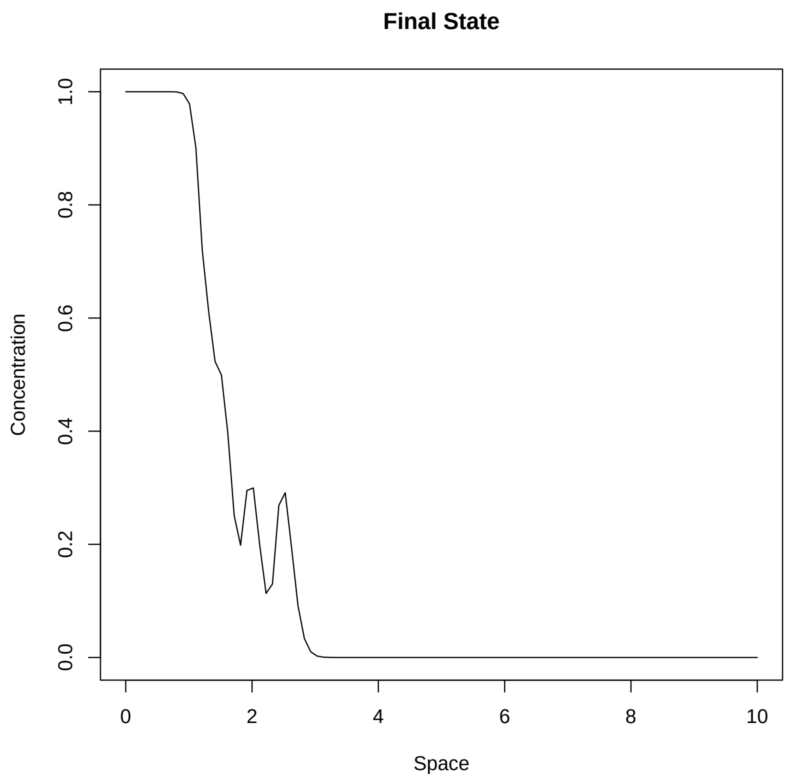

Time = 0.7
Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
Space


Time = 0.8
Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
Space


Time = 0.9
Concentration
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
Space




```
# Plot the final state
plot(seq(0, L, length.out = Nx), u, type = "l", ylim = c(0, 1), xlab = "Space", ylab = "Concentration", ma
```



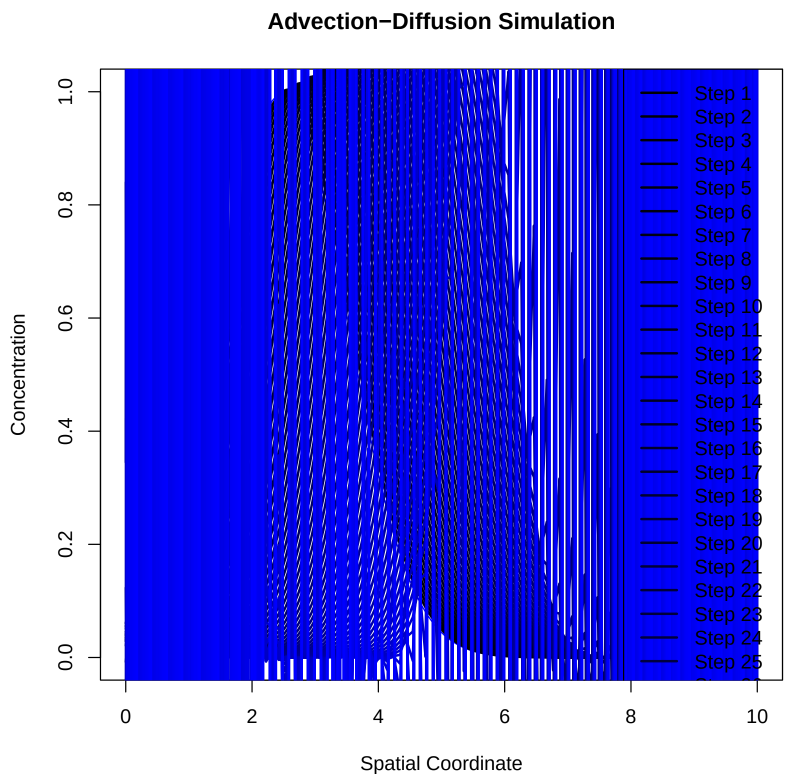
This code sets up a simple 1D advection-diffusion simulation with a pulse as the initial condition. The finite difference method is used to update the concentration at each spatial point over time. The simulation progresses in time steps, and the final state is plotted. You can adjust the parameters (e.g., diffusion coefficient, advection coefficient, grid size, time steps) to see how they affect the simulation.

Modeling Advection and Diffusion

Below is an example of R code for simulating advection and diffusion using a simple finite difference method. This code assumes a one-dimensional domain for simplicity.

```
# Advection-Diffusion Modeling in R

# Parameters
length_domain <- 10  # Length of the domain
num_points <- 100    # Number of spatial points
dx <- length_domain / (num_points - 1)  # Spatial grid size
dt <- 0.1            # Time step
```

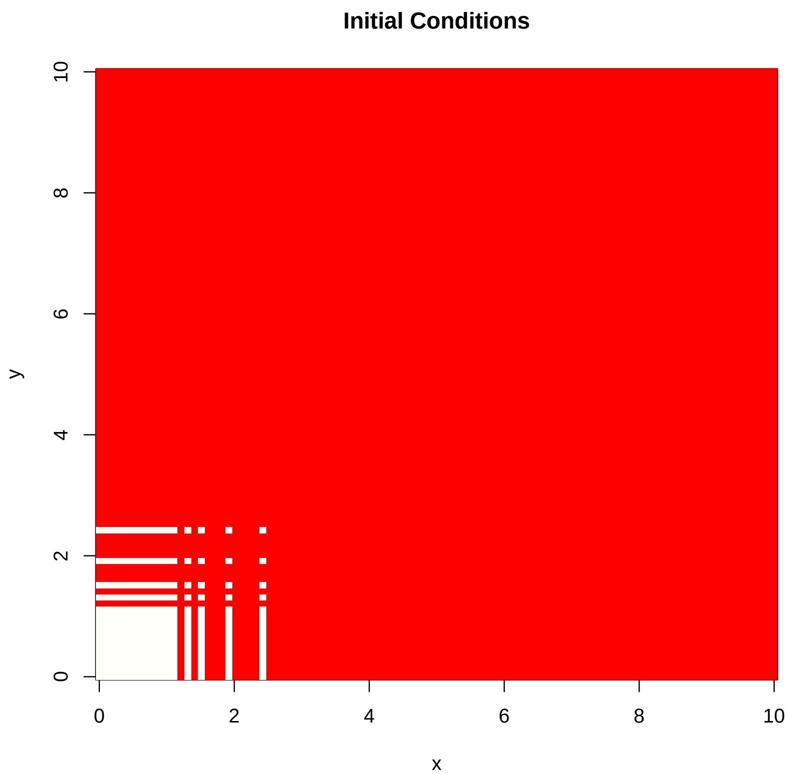
```
num_steps <- 100     # Number of time steps

# Advection and diffusion coefficients
velocity <- 0.5      # Advection velocity
diffusion_coeff <- 0.01  # Diffusion coefficient

# Initial condition
initial_condition <- function(x) {
  return(exp(-((x - length_domain/4)^2)/(2*1^2)))
}

# Initialize the concentration field
concentration <- initial_condition(seq(0, length_domain, by = dx))

# Plot initial condition
plot(seq(0, length_domain, by = dx), concentration, type = 'l', col = 'blue',
     ylim = c(0, 1), main = 'Advection-Diffusion Simulation',
     xlab = 'Spatial Coordinate', ylab = 'Concentration')

# Simulation loop
for (step in 1:num_steps) {
  # Advection term
  advected_concentration <- c(concentration[-1], concentration[1])

  # Diffusion term
  diffused_concentration <- diffusion_coeff * (c(concentration[2:num_points], concentration[num_points]) -

  # Update concentration using finite difference method
  concentration <- concentration - velocity * (advected_concentration - concentration) * dt / dx + diffuse

  # Plot the updated concentration
  lines(seq(0, length_domain, by = dx), concentration, col = rgb(0, 0, step/num_steps), lwd = 2)
}

# Add a legend
legend('topright', legend = paste('Step', 1:num_steps), col = rgb(0, 0, 1:num_steps, maxColorValue = num_s
```



This code defines a one-dimensional domain and simulates the advection and diffusion of a concentration field over time using a finite difference method. The initial condition is a Gaussian-shaped concentration profile, and the simulation updates the concentration based on advection and diffusion at each time step. The resulting animation shows the evolution of the concentration profile over time. You can adjust the parameters (e.g., diffusion coefficient, advection coefficient, grid size, time steps) to see how they affect the simulation.

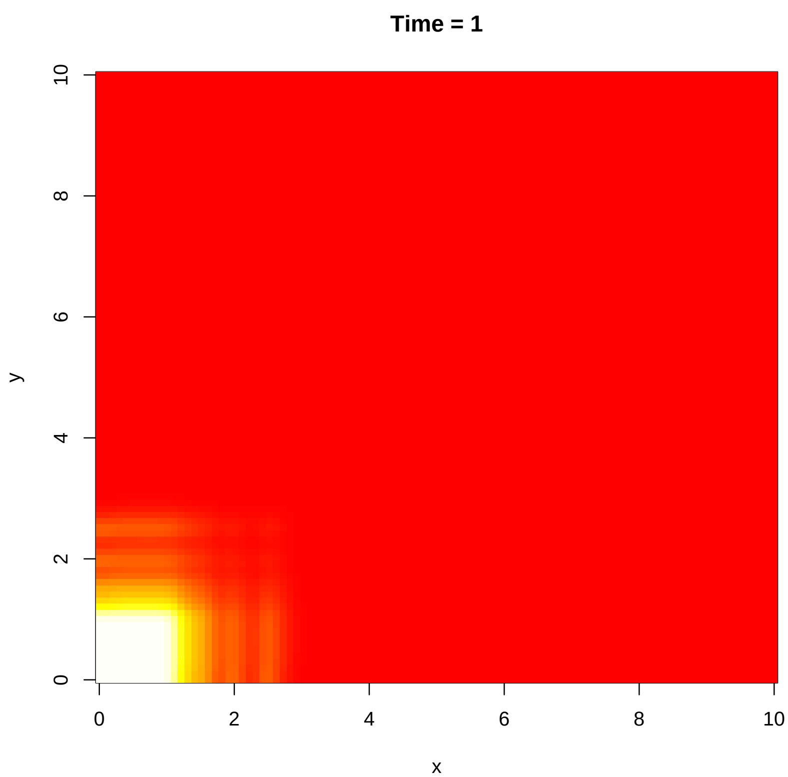
```
# Model 2 dimensional advection-diffusion using finite difference method

## set up boundaries
L <- 10
Nx <- 100
Ny <- 100
dx <- L / (Nx - 1)
dy <- L / (Ny - 1)
dt <- 0.1
Nt <- 100
```

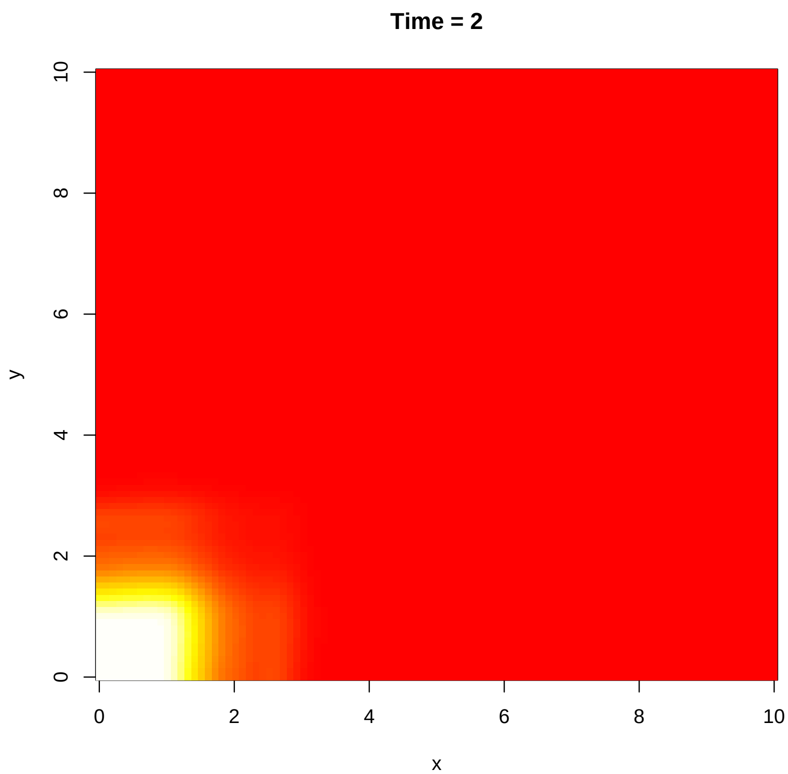
```
## set up initial conditions
u <- matrix(0, nrow = Nx, ncol = Ny)
u[Nx %/% 4:(3 * Nx %/% 4), Ny %/% 4:(3 * Ny %/% 4)] <- 1

## set up coefficients
alpha <- 0.01
beta <- 0.1

## plot initial conditions
image(seq(0, L, length.out = Nx), seq(0, L, length.out = Ny), u, col = heat.colors(100), xlab = "x", ylab
```

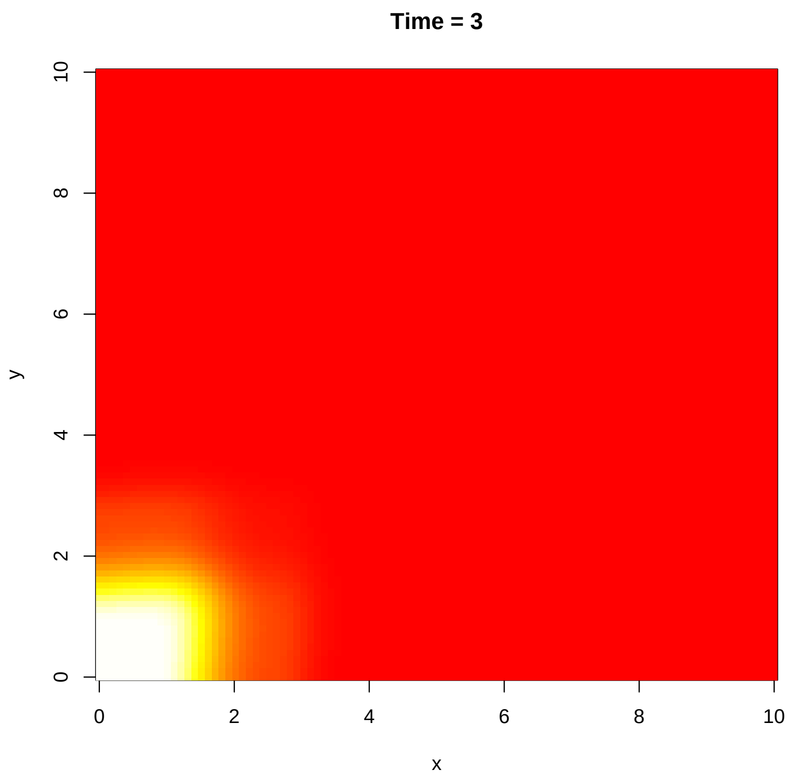


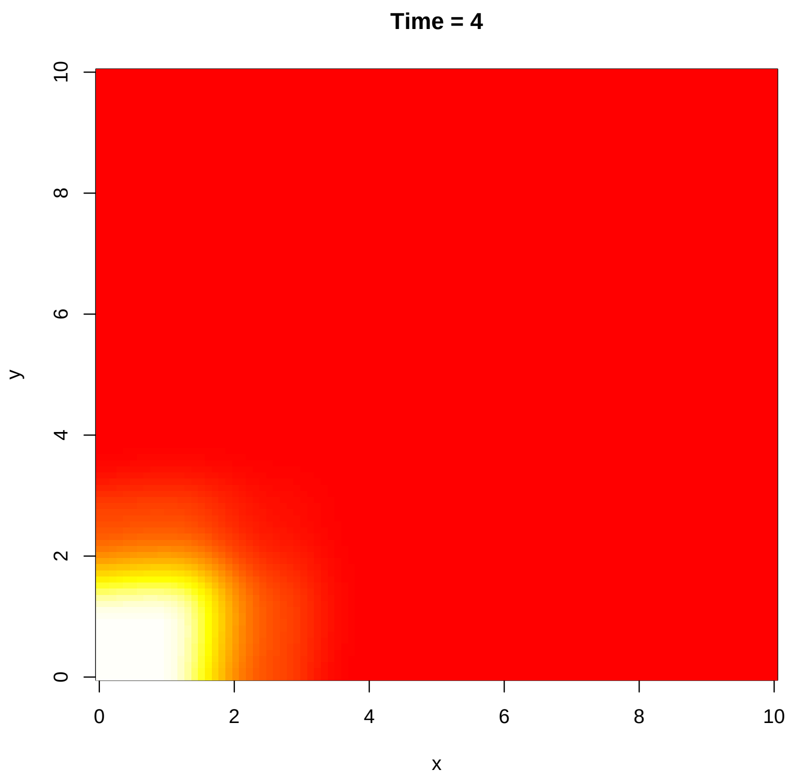

```
## simulation loop
for (t in 1:Nt) {
  for (x in 2:(Nx - 1)) {
    for (y in 2:(Ny - 1)) {
      u[x, y] <- u[x, y] + alpha * (u[x + 1, y] - 2 * u[x, y] + u[x - 1, y]) * dt / dx^2 + alpha * (u[x, y
    }
  }

  ## apply boundary conditions (zero-flux)
  u[1, ] <- u[2, ]
```

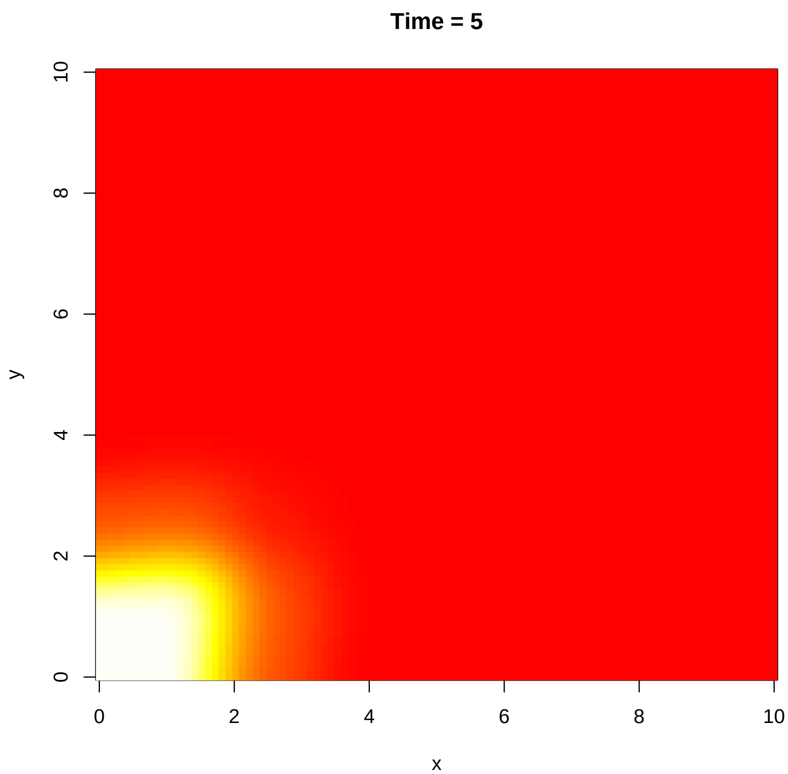
```
  u[Nx, ] <- u[Nx - 1, ]
  u[, 1] <- u[, 2]
  u[, Ny] <- u[, Ny - 1]

  ## plot the current state every 10 time steps
  if (t %% 10 == 0) {
    image(seq(0, L, length.out = Nx), seq(0, L, length.out = Ny), u, col = heat.colors(100), xlab = "x", y
    Sys.sleep(0.1)  # pause to visualize the animation
  }
}
```

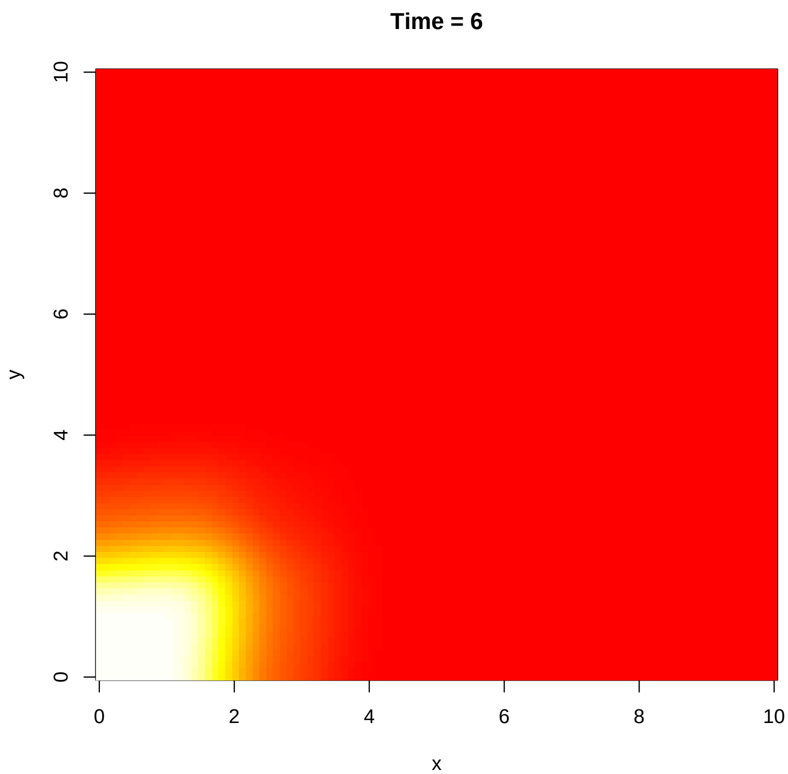


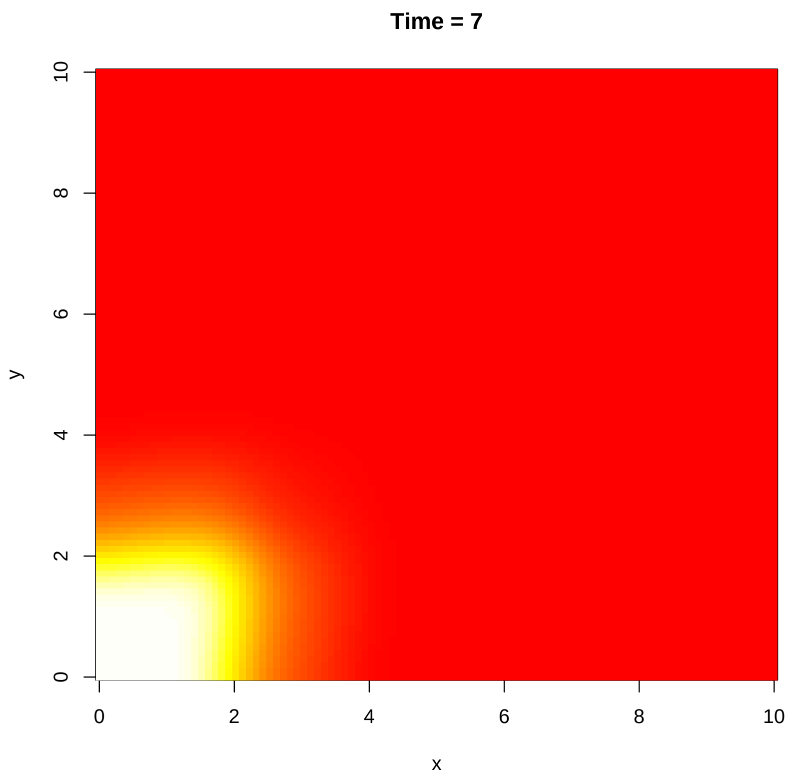

Time = 2
y
x
0
2
4
6
8
10
0
2
4
6
8
10

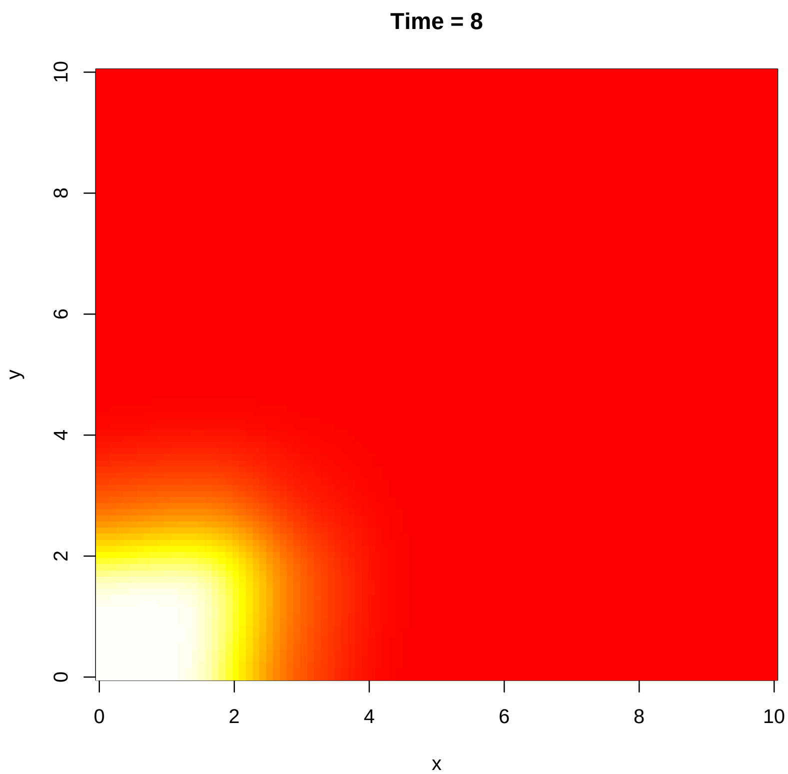

Time = 3
0
2
4
6
8
10
x
y

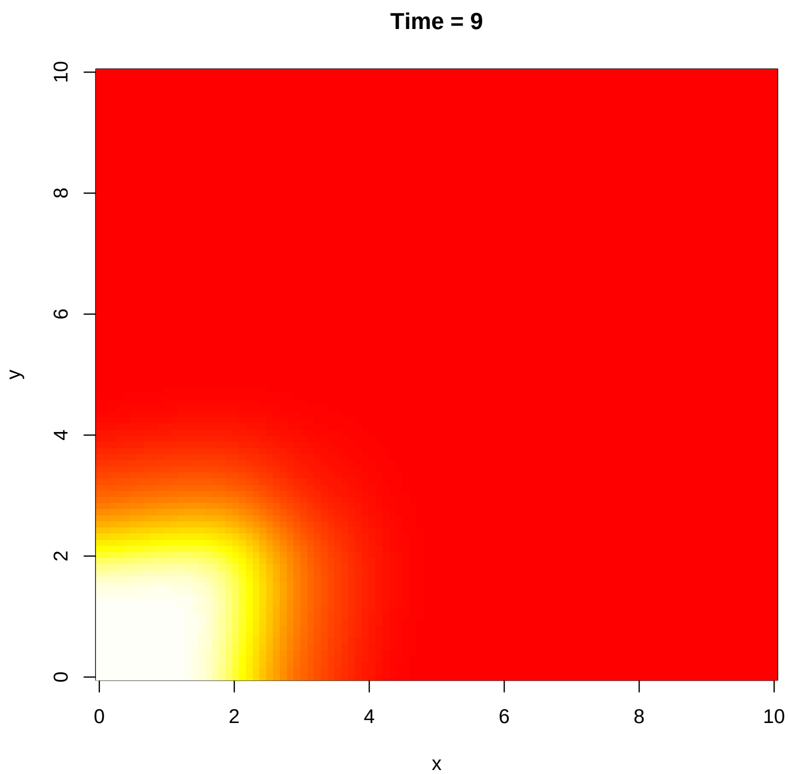

Time = 4
y
x
0
2
4
6
8
10
0
2
4
6
8
10

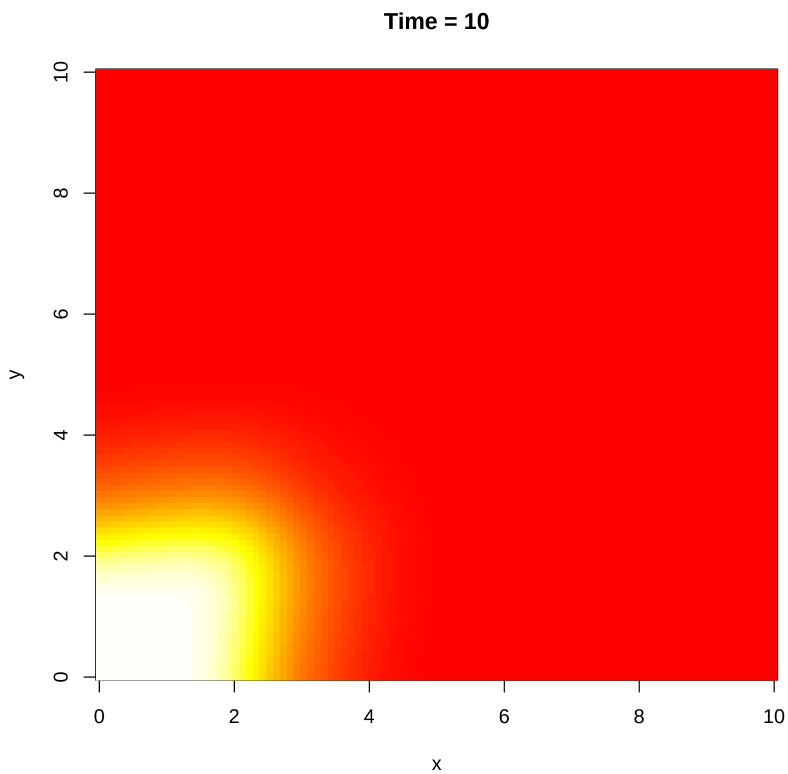

Time = 5
10
8
6
4
2
0
y
0
2
4
6
8
10
x


Time = 6
y
x
0
2
4
6
8
10
0
2
4
6
8
10

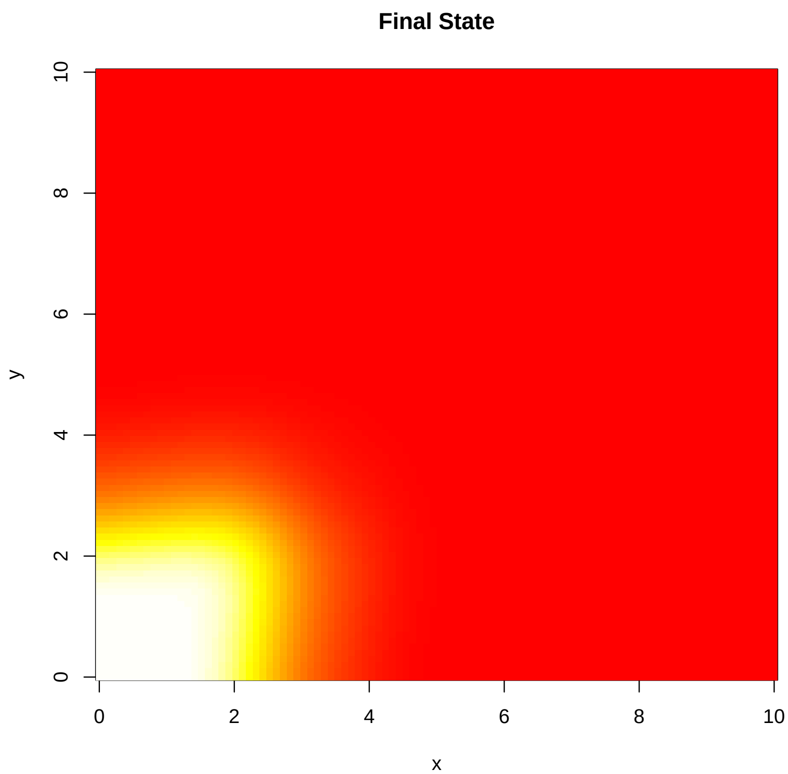

Time = 7
y
10
8
6
4
2
0
0
2
4
6
8
10
x


Time = 8
0
2
4
6
8
10
y
0
2
4
6
8
10
x


Time = 9
y
x
0
2
4
6
8
10
0
2
4
6
8
10




```
## plot the final state
image(seq(0, L, length.out = Nx), seq(0, L, length.out = Ny), u, col = heat.colors(100), xlab = "x", ylab
```


Final State
y
0
2
4
6
8
10
x
0
2
4
6
8
10