Math 586 Finally Project
By Marvyn Bailly (Github: MarvynB)

Problem 1 *Consider the ODE:*

$$v'(t) = \frac{1}{\epsilon} g(v(t)), \quad g(v) = v(\alpha - v)(v - 1) - w.$$

Using the TR-BDF-2 method,

$$U^* = U^n + \frac{k}{4}(f(U^n) + f(U^*)), U^{n+1} = \frac{1}{3}\left(4U^* - U^n + kf(U^{n+1})\right),$$

solve this ODE and verify that the method is indeed second-order accurate at $t = 0.1$. *Use* $v(0) = 0.3$, $\alpha = 0.5$, $w = 0.1$ *and* $\epsilon = 0.01$ *as your parameters.*

Note that you have to solve two nonlinear systems at each time step. The first one requires you to solve

$$G_1(u) = 0, \quad G_1(u) = u - U^n - \frac{k}{4}(f(U^n) + f(u)), \quad G_1'(u) = 1 - \frac{k}{4} f'(u).$$

The second is given by

$$G_2(u) = 0, \quad G_2(u) = u - \frac{1}{3}\left(4U^* - U^n + kf(u)\right), \quad G_2'(u) = 1 - \frac{k}{3} f'(u).$$

It is highly recommended that your write a function `TRBDF2(U,w,k)` *that advances the solution one time step. This function may take in the* `max_iter` *and* `tol` *for Newton's method as well.*

Solution.
Consider the ODE:

$$v'(t) = \frac{1}{\epsilon} g(v(t)), \quad g(v) = v(\alpha - v)(v - 1) - w.$$

Using the TR-BDF-2 method,

$$U^* = U^n + \frac{k}{4}(f(U^n) + f(U^*)), U^{n+1} = \frac{1}{3}\left(4U^* - U^n + kf(U^{n+1})\right), \tag{1}$$

we wish to solve this ODE and verify that the method is indeed second-order accurate at $t = 0.1$. We will use $v(0) = 0.3, \alpha = 0.5, w = 0.1,$ and $, \epsilon = 0.01$ as the parameters. We will write a function `TRBDF2(U,w,k,max_iter,tol)` that applies one step of the method to the solution. We implement this function in Julia as following

```
function TRBDF2(U,w,k,max_iter,tol)
    function G1(u)
        u - U - k/4 * (f(U) + f(u))
    end
    function G1_prime(u)
        1 - k/4 * f_prime(u)
    end
    function G2(u)
        u - 1/3 * (4*Us - U + k*f(u))
    end
    function G2_prime(u)
        1 - k/3 * f_prime(u)
    end
    function g(u)
        u*(α - u)*(u-1)-w
    end

    function f(x)
        (1/ϵ) * g(x)
    end

    function f_prime(u)
        (1/ϵ)*(2 * u * α - α - 3*u^2 + 2*u)
    end

    Us = Newton(U,G1,G1_prime,300,1e-10)
    U = Newton(Us,G2,G2_prime,300,1e-10)
end
```

which depends on the following implementation of Newton's method

```
function Newton(U,f,Df,max_iter,tol)
    for j=1:max_iter
        U = U .- f(U)./Df(U)
        if maximum(abs.(f(U))) < tol
            break
        end
    end
    return U
end
```

Now to see if the method is working, we plot the evolution of the solution using $k = 0.001$ from $t = 0$ to $t = 0.1$ to get the plot shown in Figure 1 which appears to be demonstrating the correct behavior of the solution. Next, we will numerically show that the method is second-order accurate at $t = 0.1$ by creating a reference solution using a small step size of $k = 2^{-10}t$. Next, we compute the solution at $t = 0.1$ using increasingly small k values, $k = 2^(-j)t$ for $j = 1, 2, \cdots, 8$ and computed the maximum absolute error from the reference solution. Plotting the maximum errors along with $y = h^2$ we see the errors decrease with $y = h^2$ as seen in Figure 2. Therefore we conclude that this method is second-order accurate.

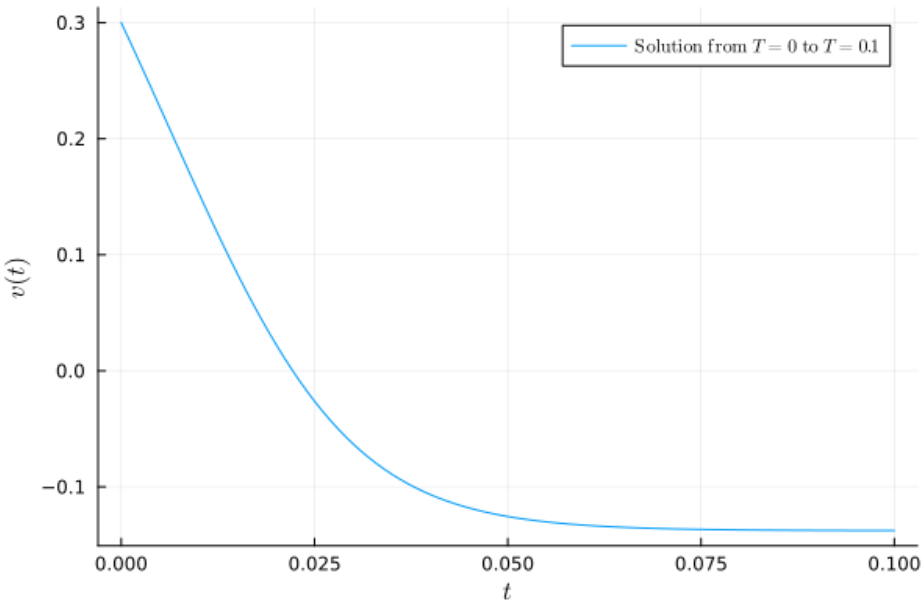


Figure 1: Evolution of solution from $t = 0$ to $t = 0.1$ using $k = 0.001$.

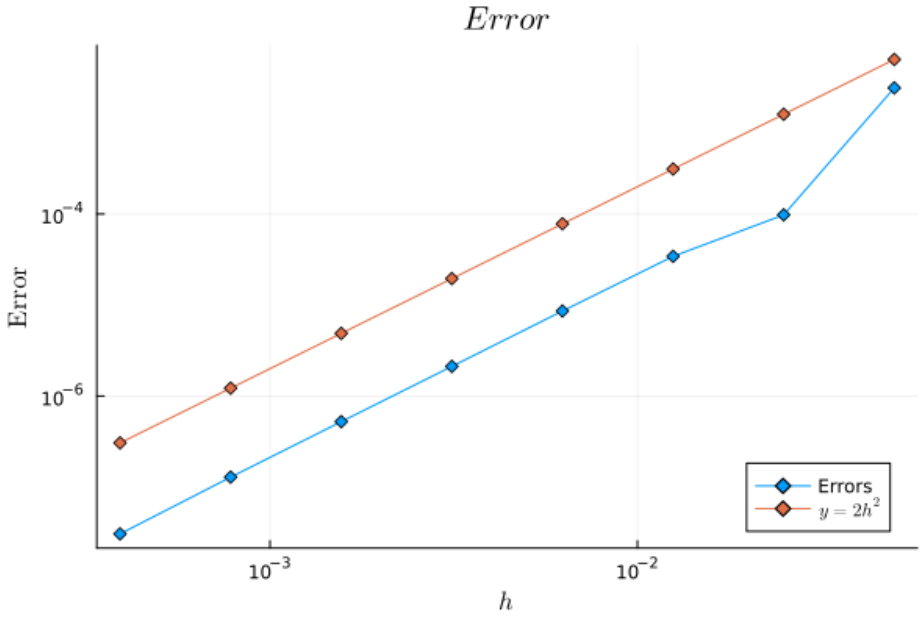


Figure 2: Maximum errors, seen in blue, between reference solution, $k = T2^{-10}$, and increasingly small k values at $t = 0.1$. In red $y = 2h^2$.

□

Problem 2 *The previous ODE becomes more interesting when it is coupled with another differential equation that evolves w:*

$$v'(t) = \frac{1}{\epsilon}\left(g(v(t)) - w(t) + I_a\right), \quad g(v) = v(\alpha - v)(v-1),$$
$$w'(t) = \beta v(t) - \gamma w(t).$$

This is a simple model that exhibits many features of excitable media, and is a simplification of the famous Hodgkin-Huxley equations that are a model for propagation in neuronal systems. In this system $v(t)$ models the membrane potential while $w(t)$ models the concentration of an ion. Since we have already gone through the effort to solve the above system, we would like to be able to reuse our code. Write the system as

$$\begin{bmatrix} v'(t) \\ w'(t) \end{bmatrix} = \mathcal{A}\left(\begin{bmatrix} v(t) \\ w(t) \end{bmatrix}\right) + \mathcal{B}\left(\begin{bmatrix} v(t) \\ w(t) \end{bmatrix}\right)$$

where

$$\mathcal{A}\left(\begin{bmatrix} v \\ w \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{\epsilon}\left(g(v) - w + I_a\right) \\ 0 \end{bmatrix},$$
$$\mathcal{B}\left(\begin{bmatrix} v \\ w \end{bmatrix}\right) = \begin{bmatrix} 0 \\ \beta v - \gamma w \end{bmatrix}.$$

In the notation of (11.24) in the text, the function `TRBDF2(V,W,k)` *described in the previous problem is the function $\mathcal{N}_{\mathcal{A}}(V, W, k)$. Construct the analogous function for $\mathcal{N}_{\mathcal{B}}(V, W, k)$, call it* `RK2(V,W)` *using the second-order Runge-Kutta method (5.30). Then* `Julia` *code for Strang splitting (11.24) is given by*

```
W[i] = RK2(V[i-1],W[i-1],k/2)
V[i] = TRBDF2(V[i-1],W[i],k)
W[i] = RK2(V[i],W[i],k/2)
```

This should be easily adapted to the language of your choosing. Using

$$\alpha = 0.3, \quad \beta = 1, \quad \gamma = 1, \quad I_a = 0, \quad \epsilon = 0.001,$$

with initial data

$$v(0) = v_0, \quad w(0) = 0,$$

the solution of the ODE system exhibits two distinct behaviors:

1. If $v_0 = 0.31$ then $v(t)$ rapdily rises near $v = 1$ and then decays to the stable steady state $v = w = 0$.

2. If $v_0 = 0.29$ the the solution decays directly to the steady state $v = w = 0$. Verify the plot these two solutions. Numerically, verify that your method is second-order accurate at $t = 0.25$ when $v_0 = 0.29$. Lastly, change $I_a = 0.2$ and plot the solution with $v_0 = 0$ until $t = 10$.

Solution.
Consider modifying the previous ODE such that it is coupled with another differential equation that evolves w:

$$v'(t) = \frac{1}{\epsilon}\left(g(v(t)) - w(t) + I_a\right), \quad g(v) = v(\alpha - v)(v-1),$$
$$w'(t) = \beta v(t) - \gamma w(t).$$

We will rewrite this system as

$$\begin{bmatrix} v'(t) \\ w'(t) \end{bmatrix} = \mathcal{A}\left(\begin{bmatrix} v(t) \\ w(t) \end{bmatrix}\right) + \mathcal{B}\left(\begin{bmatrix} v(t) \\ w(t) \end{bmatrix}\right)$$

where

$$\mathcal{A}\left(\begin{bmatrix} v \\ w \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{\epsilon}\left(g(v) - w + I_a\right) \\ 0 \end{bmatrix},$$
$$\mathcal{B}\left(\begin{bmatrix} v \\ w \end{bmatrix}\right) = \begin{bmatrix} 0 \\ \beta v - \gamma w \end{bmatrix}.$$

We will use a Strang splitting method to solve this problem by splitting into $\mathcal{N}_{\mathcal{A}}(V, W, k)$ and $\mathcal{N}_{\mathcal{B}}(V, W, k)$. To solve $\mathcal{N}_{\mathcal{A}}(V, W, k)$ we will modify the previous `TRBDF2(U,w,k,max_iter,tol)` to fit the updated $v'(t)$ function as seen in the following code:

```
function TRBDF2(U,w,k,max_iter,tol)
    function G1(u)
        u .- U .- k/4 .* (f(U) .+ f(u))
    end

    function G1_prime(u)
        1 .- k/4 .* f_prime(u)
    end

    function G2(u)
        u .- 1/3 .* (4 .* Us .- U .+ k .* f(u))
    end

    function G2_prime(u)
        1 .- k/3 .* f_prime(u)
    end
    function g(v)
        v .* (α .- v) .* (v .- 1)
    end

    function f(x)
        (1 ./ ϵ) .* (g(x) .- w .+ Ia)
    end

    function f_prime(u)
        (1 ./ ϵ) .* (2 .* u .* α .- α .- 3 .* u^2 .+ 2 .* u)
    end

    Us = Newton(U,G1,G1_prime,300,1e-10)
    U = Newton(Us,G2,G2_prime,300,1e-10)
end
```

To handle $\mathcal{N}_{\mathcal{B}}(V, W, k)$, we will use the second-order Runge-Kutta method and call the function `RK2(V,W)` which is implemented in the following code:

```
function RK2(V,W,k)
    function h(W,V)
        β.*V .- γ.*W
    end
    Wtemp = W .+ (k/2) .* h(W,V)
    W = W .+ k .* h(Wtemp,V)
end
```

The Strang splitting method we use is given by

```
W[i] = RK2(V[i-1],W[i-1],k/2)
V[i] = TRBDF2(V[i-1],W[i],k,300,1e-10)
W[i] = RK2(V[i],W[i],k/2)
```

We will use the parameters $\alpha = 0.3, \beta = 1, \gamma = 1, I_a = 0$, and $\epsilon = 0.001$ with the initial conditions $v(0) = v_0, \quad w(0) = 0$.
We will first plot the evolution of the solution for $v_0 = 0.31$ on $t = 0$ to $t = 1.5$ which can be seen in Figure 3. We see that $v(t)$ rapidly rises near $v(t) = 1$ and then decays to the stable steady state of $v(t) = w(t) = 0$. We also see that $w(t)$ increases until it intersects $v(t)$ and then decreases to the stable steady state.
Next, we will plot the evolution of the solution for $v_0 = 0.29$ on the $t = 0$ to $t = 0.25$ which can be seen in Figure 4. We see that solution $v(t)$ decays directly to the steady state and once again it appears that $w(t)$ slightly increases until it intersects $v(t)$ and then decreases to the steady state.
Next, we will numerically verify that our method is second-order accurate at $t = 0.25$ when $v_0 = 0.29$. We will use the same method as described in question 1 by creating a reference solution using a small k value of $k = 2^{-16}t$. We will run the solution for increasingly small k values, $k = t2^{-j}$ for $j = 1, 2, \cdots, 8$ and compute the maximum absolute error of $v(t)$ and $w(t)$ compared to the reference solution for $v(t)$ and $w(t)$. Then plotting the maximum of the max error of $v(t)$ and $w(t)$ at each k step along with $y = h^2$, we see that the errors decrease with $y = h^2$ as seen in Figure 2. Therefore we conclude that this method is second-order accurate.
Finally, we change $I_a = 0.2$ and $v_0 = 0$ and plot the evolution of the solution from $t = 0$ to $t = 10$. We see the solution in Figure 6 where we notice the relationship between $v(t)$ and $w(t)$. As $v(t)$ spikes to around $v(t) = 1$, it decreases until it spikes again. At the same time, $w(t)$ increases until it intersects $v(t)$ and decreases until it intersects $v(t)$ where it increases again.

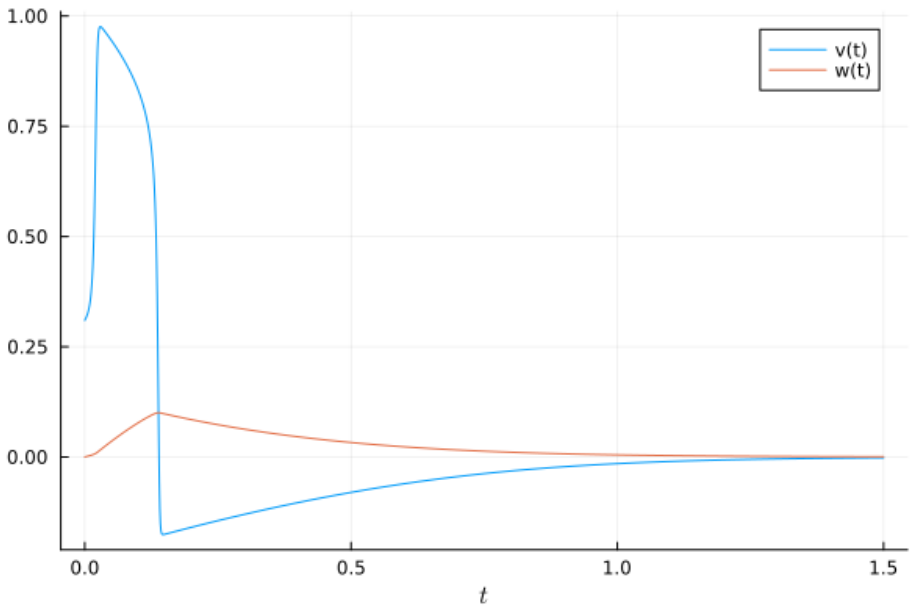


Figure 3: Solution from $t = 0$ to $t = 1.5$ with $v_0 = 0.31$. $v(t)$ seen in blue while $w(t)$ is seen in orange.

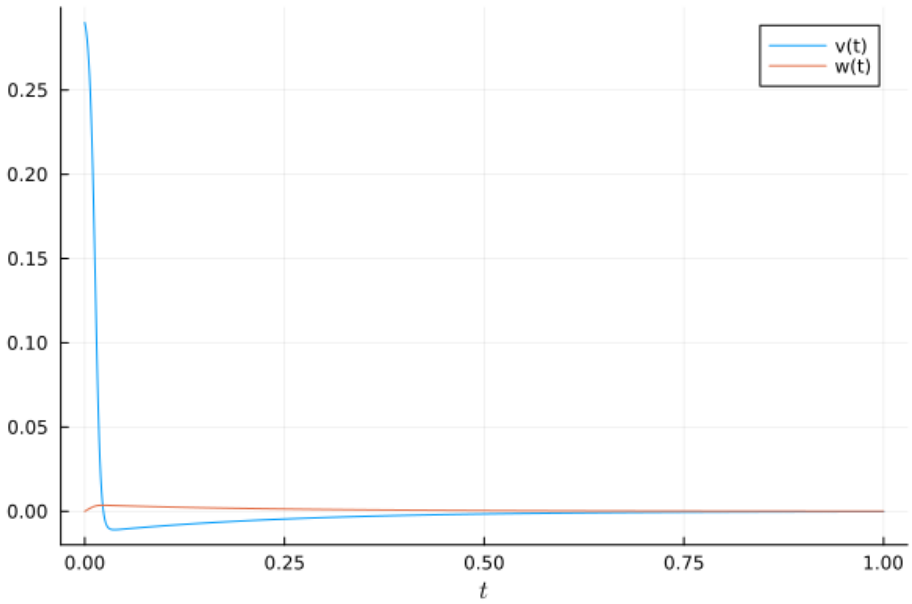


Figure 4: Solution from $t = 0$ to $t = 1.5$ with $v_0 = 0.29$. $v(t)$ seen in blue while $w(t)$ is seen in orange.

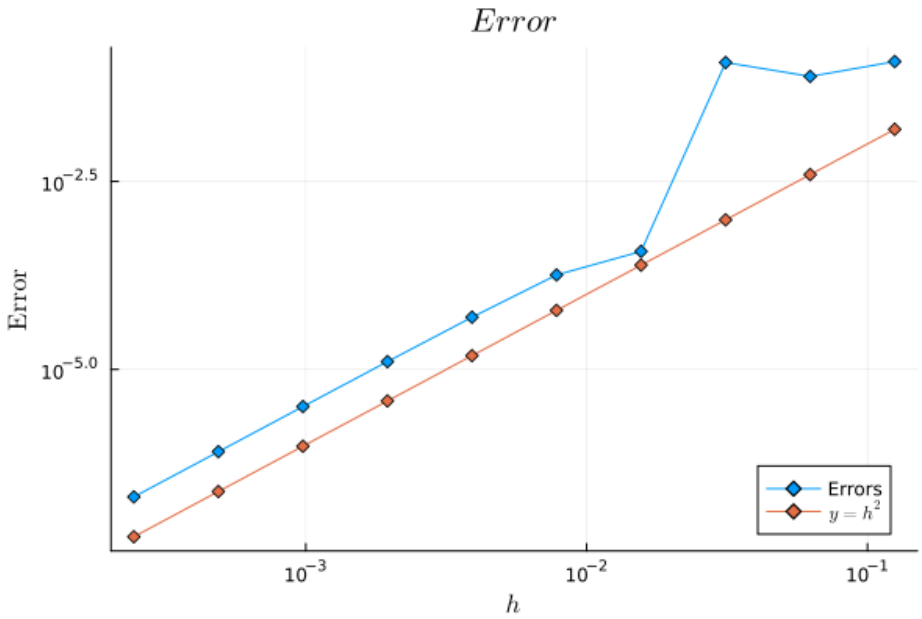


Figure 5: Plot of the max of the maximum absolute error of $v(t)$ and $w(t)$ compared to the reference solution.

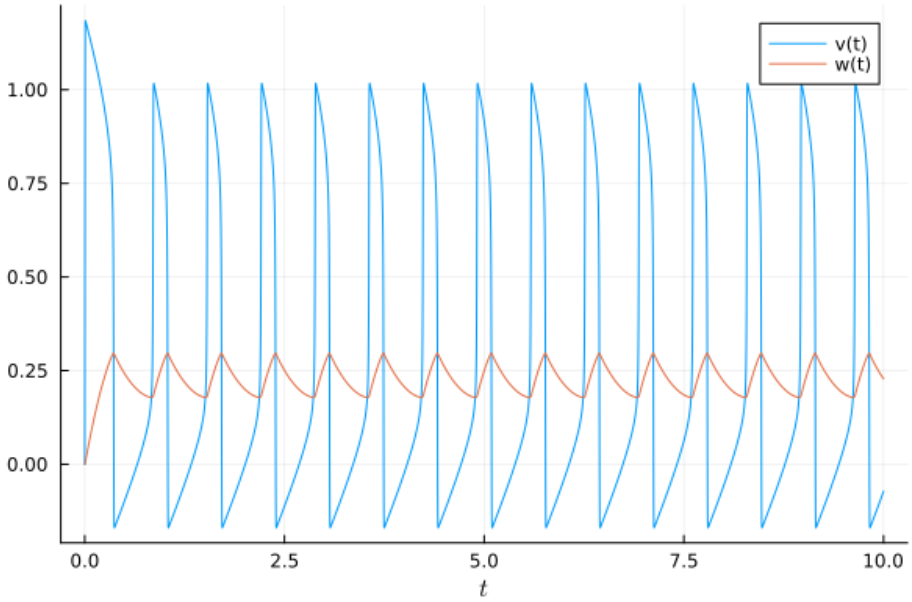


Figure 6: Solution when $I_a = 0.2$ and t goes from 0 to 10.

□

Problem 3 *Consider the heat equation*

$$v_t = \kappa v_{xx}, \quad t > 0, \quad x \in (a, b).$$

We impose Neumann boundary conditions $v_x(a,t) = 0 = v_x(b,t)$. *Because the boundary conditions do not directly specify* $v(a,t)$ *and* $v(b,t)$, *we need to include their approximations in our vector of unkowns. The MOL discretization can be written as*

$$V'(t) = \frac{\kappa}{h^2} A V(t),$$

where

$$V(t) = \begin{bmatrix} V_0(t) \\ V_1(t) \\ \vdots \\ V_{m+1}(t) \end{bmatrix},$$

and

$$A = \begin{bmatrix} -2 & 2 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 & \\ & & & & 2 & -2 & \end{bmatrix}.$$

Note: This is derived by introducing "ghost values" $V_{-1}(t)$ *and* $V_{m+2}(t)$ *and then enforcing the Neumann conditions via*

$$\frac{V_1(t) - V_{-1}(t)}{2h} = 0 = \frac{V_{m+2}(t) - V_m(t)}{2h}.$$

These relations are used to then eliminate the ghost values from the system. You need not derive this. Apply the TR-BDF-2 method to the MOL discretization and write a function `TRBDF2_heat(V)` *that applies one time step of this method with time step* `k`. *It is best to keep* `k` *fixed here because then matrices do not have to be recomputed at each time step. With* $a = 0, b = 4$, *demonstrate that your code works by using the initial data*

$$v(x,0) = \begin{cases} 1 & a \le x < \frac{a+b}{2} \\ 0 & \text{otherwise.} \end{cases}$$

The integral of the solution should nearly be preserved, resulting in a long-time limit $v(x,t) \approx 1/2$ *as* $t \to \infty$. *Demonstrate this with* $h = 0.01$, $k = h$ *and* $\kappa = 1$ *and discuss any discrepancy you see. It may help to choose other initial data to explain the situation. Note: Here you are working on an interval* $[a, b]$ *and in this case* h *is affected by the difference* $b - a$.

Solution.
Consider the heat equation

$$v_t = \kappa v_{xx}, \quad t > 0, \quad x \in (a, b),$$

with Neumann boundary conditions $v_x(a,t) = 0 = v_x(b,t)$. Because the boundary conditions do not directly specify $v(a,t)$ and $v(b,t)$, we need to include their approximations in our vector of unknowns. The MOL discretization can be written as

$$V'(t) = \frac{\kappa}{h^2} A V(t),$$

where

$$V(t) = \begin{bmatrix} V_0(t) \\ V_1(t) \\ \vdots \\ V_{m+1}(t) \end{bmatrix},$$

and

$$A = \begin{bmatrix} -2 & 2 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 2 & -2 \end{bmatrix}.$$

We wish to use our TRBDF-2 method for the MOL discretization and write a function `TRBDF2_heat(V)` that applies one time step of this method. Noticing that we can rewrite Eq. (1) as

$$\left(I - \frac{k}{4}A\right) U^* = \left(I + \frac{k}{4}A\right) U^n,$$
$$\left(I - \frac{k}{3}A\right) U^{n+1} = \frac{1}{3}\left(4U^* - U^n\right),$$

we implement the `TRBDF_heat(v)`' as

```
function TRBDF2_heat(U,A1,A2,A3)
    Us = A1 \ (A2 * U)
    return A3\ ((4/3) * Us - (1/3) * U)
end
```

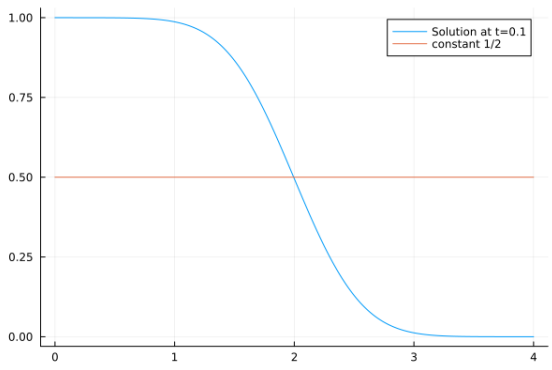
where $A_1 = I - \frac{k}{4}A$, $A_2 = I + \frac{k}{4}A$, and $A_3 = I - \frac{k}{3}A$. Note that as per the recommendation, we define this function for a fixed k to avoid the computation of the matrices on each step. We let $a = 0, b = 4, h = 0.01, k = h, \kappa = 1$ and use the initial condition

$$\eta(x) = v(x, 0) = \begin{cases} 1 & a \leq x < \frac{a+b}{2} \\ 0 & \text{otherwise.} \end{cases}$$

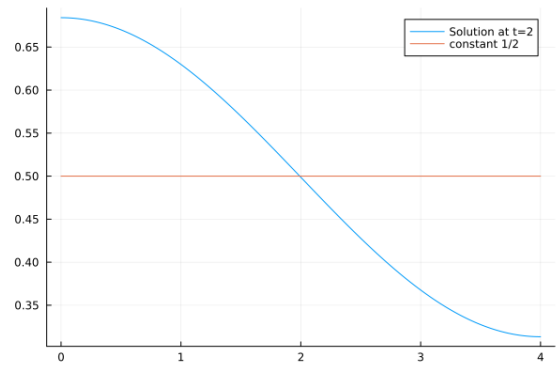
To demonstrate that our method is working correctly, we note that the integral of the solution should nearly be preserved, resulting in a long-time limit $v(x, t) \approx 1/2$ as $t \to \infty$. We create a nice gif in the Notebook (see src/code/3.gif). Plotting the solution with the initial condition at $t = 0.1, 2, 5, 10$ we see in Figure 7 that the $v(x, t) \approx 1/2$ as t gets big as desired. But if we take a closer look at the solution as t gets large, we see that the solution is settling at a position that is pretty far away from $v(x, t) = 1/2$ as seen in Figure. Computing the maximum error between the solution and $1/2$ we get an error of 0.0012527925659450156 which is pretty far off from $1/2$. If we try an initial condition such as

$$\eta_1(x) = \cos\left(\frac{\pi}{4}x\right),$$

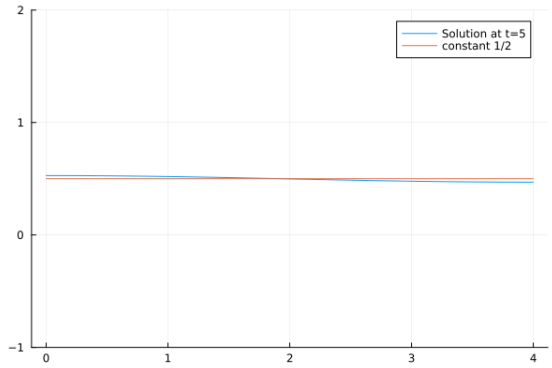
we expect the method to be able to handle the initial condition since it is a solution to the PDE and expect $v(x, t) \to 0$ as $t \to \infty$. When computing the maximum error between the solution at $t = 100$ and 0 we get an error of $1.807789142202125e - 13$. This could show a slight discrepancy in our numerical method.



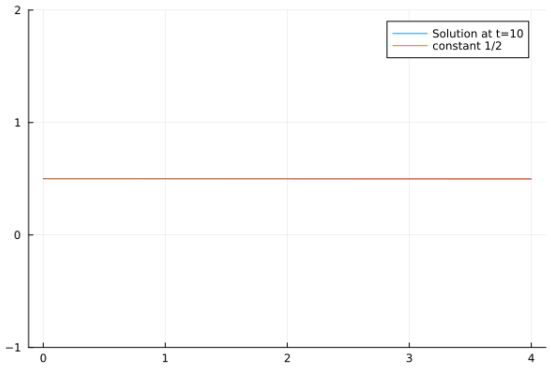
(a) Solution at $t = 0.1$.



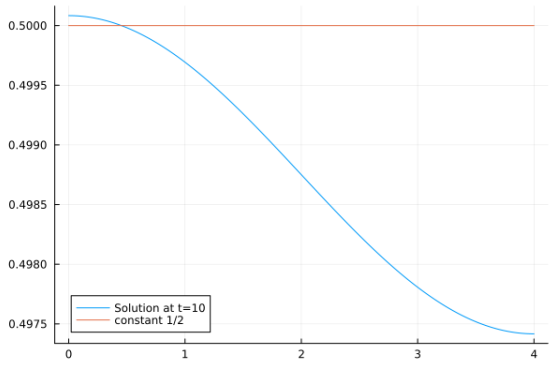
(b) Solution at $t = 2.0$.



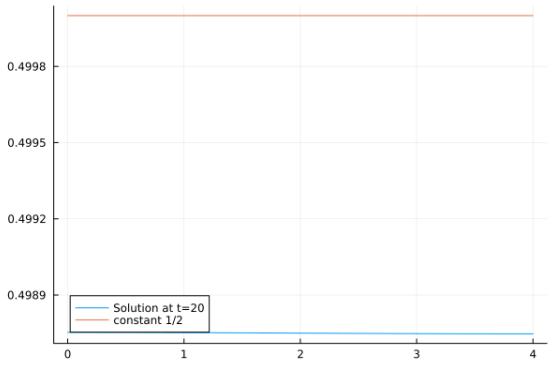
(c) Solution at $t = 5.0$.



(d) Solution at $t = 10.0$.



(e) A closer look at solution at $t = 10.0$.



(f) Solution at $t = 20.0$.

Figure 7: Solution using the initial condition $\eta(x)$ at $t = 0.1, 2.0, 5.0, 10.0,$ and 20.0. The solution is shown in blue while a constant $v = 1/2$ is shown in orange.

□

Problem 4 *The ODE system can be modified to allow for spatial variations: For* $x \in (a, b)$

$$\begin{aligned} v_t(x,t) &= \kappa v_{xx}(x,t) + \frac{1}{\epsilon}\left(g(v(x,t)) - w(x,t) + I_a\right), \quad g(v) = v(\alpha - v)(v-1), \\ w_t(x,t) &= \beta v(x,t) - \gamma w(x,t) \\ v_x(a,t) &= 0, \\ v_x(b,t) &= 0. \end{aligned}$$

Now let function `TRBDF2_heat(V)` *apply one time step toward the solution of* $v_t = \kappa v_{xx}$ *with time step* `k/2`*. Supposing that our previous code for* `TRBDF2()` *and* `RK2()` *is vectorized, Strang splitting becomes (with* $I_a = 0$*)*

```
V = TRBDF2_heat(V)
W = RK2(V,W,k/2)
V = TRBDF2(V,W,k)
W = RK2(V,W,k/2)
V = TRBDF2_heat(V)
```

Use the parameters:

$$\begin{gathered} h = 0.02, \quad k = h/10, \quad a = 0, \quad b = 6, \quad \alpha = 0.3, \\ \beta = 1, \quad \gamma = 1, \quad \kappa = 0.2, \quad \epsilon = 0.001. \end{gathered}$$

and initial data

$$v(x,0) = \begin{cases} 1 & 0 \le x < 1 \\ 0 & otherwise, \end{cases} \qquad w(x,0) = 0.$$

Solve until $t = 4$*. You should see a travelling wave develop and propagate.*

Solution. Consider the modified ODE system which allows for spatial variations: For $x \in (a, b)$

$$\begin{aligned} v_t(x,t) &= \kappa v_{xx}(x,t) + \frac{1}{\epsilon}\left(g(v(x,t)) - w(x,t) + I_a\right), \quad g(v) = v(\alpha - v)(v-1), \\ w_t(x,t) &= \beta v(x,t) - \gamma w(x,t) \\ v_x(a,t) &= 0, \\ v_x(b,t) &= 0. \end{aligned}$$

We will let the function `TRBDF2_heat(V)` apply one-time step toward the solution of $v_t = \kappa v_{xx}$ with time step `k/2`. To account for the new time step, we modify the matrices $A_1, A_2,$ and A_3 to $A_1 = I - \frac{k}{8}A$, $A_2 = I + \frac{k}{8}A$, and $A_3 = I - \frac{k}{6}A$ to account for the half time step. Now using our previous code for `TRBDF2()` and `RK2()`, we apply the Strang splitting (with $I_a = 0$) of the form

```
function solve_strange(V,W,T)
    n = convert(Int64,ceil(T/k))
    t=0
    n = convert(Int64,ceil(T/k))
    for i = 2:n+1
        t += k
        #STRANG SPLITTING
        V = TRBDF2_heat(V) #V=0
        W = RK2(V,W,k/2)
        V = TRBDF2(V,W,k,300,1e-10) #V,W
        W = RK2(V,W,k/2)
        V = TRBDF2_heat(V)
    end
    return V,W
end
```

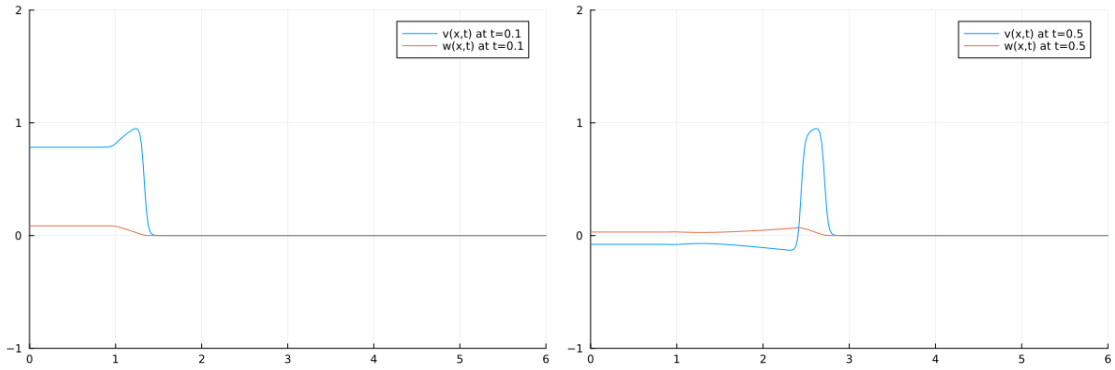
Now using the parameters:

$$h = 0.02, \quad k = h/10, \quad a = 0, \quad b = 6, \quad \alpha = 0.3,$$
$$\beta = 1, \quad \gamma = 1, \quad \kappa = 0.2, \quad \epsilon = 0.001,$$

and initial data

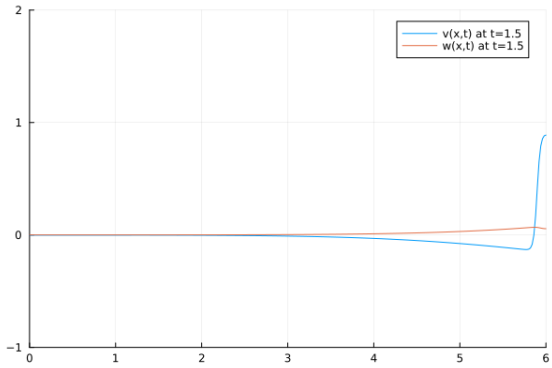
$$v(x,0) = \begin{cases} 1 & 0 \leq x < 1 \\ 0 & \text{otherwise,} \end{cases} \qquad w(x,0) = 0,$$

we solve the modified ODE system until $t = 4$. In my Jupiter notebook, we create a nice gif (see /src/code/4.gif) of the evolution of the solutions $v(x,t)$ and $w(x,t)$ from $t = 0$ to $t = 4$. In the gif and the four presented figures in Figure 8, we see a traveling wave emerge at $t = 0$ and travel to the right as time increases. It is super neat to see how the same interaction between $v(x,t)$ and $w(x,t)$ as discussed in the previous questions is present and appears to drive the traveling wave.

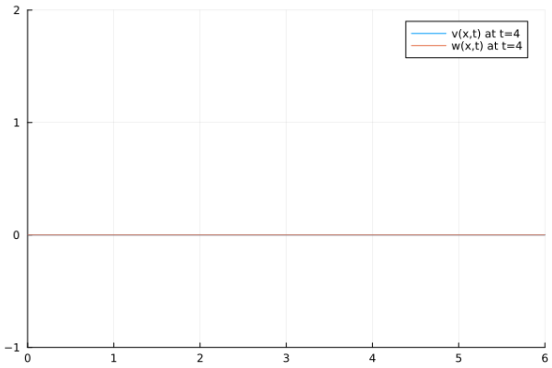


(a) Solution at $t = 0.1$.

(b) Solution at $t = 0.5$.



(c) Solution at $t = 1.5$.



(d) Solution at $t = 4.0$.

Figure 8: Solutions of the modified system of ODEs. We see $v(x,t)$ in blue and $w(x,t)$ in orange at times $t = 0.1, 0.5, 1.5,$ and 4.0

□

Problem 5 *Repeat the previous problem with*

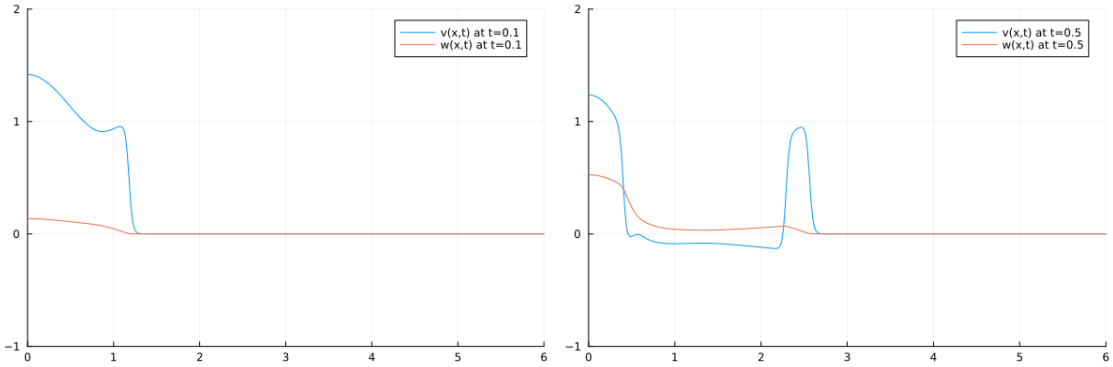
$$v(x,0) = 0, \quad w(x,0) = 0, \quad I_a(x) = 0.8e^{-5x^2}.$$

This should be a straightforward modification of the code if you've followed the outlined setup and will produce a periodic firing of traveling waves.

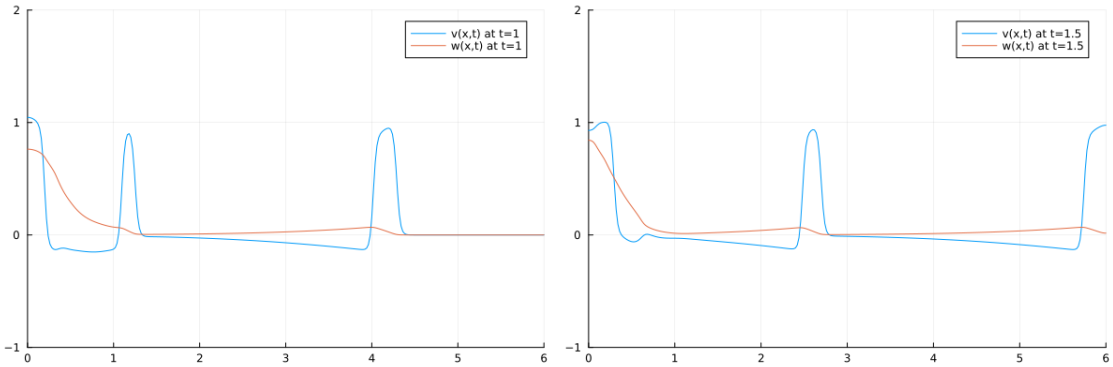
Solution.
We repeat the previous problem now using

$$v(x,0) = 0, \quad w(x,0) = 0, \quad I_a(x) = 0.8e^{-5x^2}.$$

We didn't have to make any modifications to our method code. We make a nice gif (see src/-code/5.gif)in the notebook which shows a periodic firing of traveling waves. Here we present the solutions at $t = 0.1, 0.5, 1.0,$ and 1.5 in Figure 9 which shows waves being periodically created and traveling to the right as time evolves.



(a) Solution at $t = 0.1$. (b) Solution at $t = 0.5$.



(c) Solution at $t = 1.0$. (d) Solution at $t = 1.5$.

Figure 9: Solutions of the modified system of ODEs. We see $v(x,t)$ in blue and $w(x,t)$ in orange at times $t = 0.1, 0.5, 1.0,$ and 1.5

□

Problem 6 *Consider the extension of the previous problem to two spatial dimensions: For* $x \in (a,b),\ y \in (a,b)$*:*

$$v_t(x,y,t) = \kappa v_{xx}(x,y,t) + \kappa v_{yy}(x,y,t) + \frac{1}{\epsilon}\left(g(v(x,y,t)) - w(x,y,t) + I_a\right), \quad g(v) = v(\alpha - v)(v-1),$$
$$w_t(x,y,t) = \beta v(x,y,t) - \gamma w(x,y,t).$$

with initial and boundary data given by

$$v(x,y,0) = \eta(x,y) = \begin{cases} 1 & x < 2, \\ 0 & otherwise, \end{cases} \qquad w(x,y,0) = 0,$$
$$v_x(a,y,t) = v_x(b,y,t) = v_y(x,a,t) = v_y(x,b,t) = 0.$$

The code we have created actually immediately generalizes to solve this problem. Suppose that matrices `W` *and* `V` *contain the spatial array of the values obeying the ordering in Figure 3.2(a). For example, supposing that the grid spacing is the same in both directions the following code initializes* `W` *and* `V`

```
x = a:h:b |> Array
y = x
X = repeat(reshape(x, 1, :), length(y), 1)
Y = repeat(reverse(y), 1, length(x));
η = (x,y) -> x < 2 ? 1.0 : 0.0
V = map(η,X,Y)
W = 0*V
```

Then the Strang splitting scheme is given by

```
V = TRBDF2_heat(V)
V = TRBDF2_heat(V')' |> Array
W = RK2(V,W,k/2)
V = TRBDF2(V,W,k)
W = RK2(V,W,k/2)
V = TRBDF2_heat(V)
V = TRBDF2_heat(V')' |> Array
```

The `|> Array` *call just converts the transpose to a real array.* `Julia` *has a special data type for the transpose of an array that allows for efficient memory usage but we will avoid using that here. In* `Matlab` *or* `Python` *you wouldn't need an analogous call. Set the parameters of the problem to be:*

$$a = 0, \quad b = 12, \quad h = .05, \quad k = h/10, \quad \kappa = 1, \quad \epsilon = 0.01,$$
$$\alpha = 0.1, \quad \beta = 0.5, \quad \gamma = 1, \quad I_a = 0.$$

If you just solve with these parameters you should see a single pulse travel across the domain in the positive x*-direction. But if at time* $t = 0.9$ *zero out all elements of* V *with associated* y *coordinates being larger than* 6*, you break the symmetry in the* y*-direction. Solve until* $t = 6$*. You should see a spiral wave develop!*

Solution.
Consider the extension of the previous problem to two spatial dimensions: For $x \in (a,b),\ y \in (a,b)$:

$$v_t(x,y,t) = \kappa v_{xx}(x,y,t) + \kappa v_{yy}(x,y,t) + \frac{1}{\epsilon}\left(g(v(x,y,t)) - w(x,y,t) + I_a\right), \quad g(v) = v(\alpha - v)(v-1),$$
$$w_t(x,y,t) = \beta v(x,y,t) - \gamma w(x,y,t).$$

with initial and boundary data given by

$$v(x,y,0) = \eta(x,y) = \begin{cases} 1 & x < 2, \\ 0 & \text{otherwise}, \end{cases} \qquad w(x,y,0) = 0,$$
$$v_x(a,y,t) = v_x(b,y,t) = v_y(x,a,t) = v_y(x,b,t) = 0.$$

The code we have created immediately generalizes to solve this problem. We initializes `W` and `V` as

```
x = a:h:b |> Array
y = x
X = repeat(reshape(x, 1, :), length(y), 1)
Y = repeat(reverse(y), 1, length(x));
η = (x,y) -> x < 2 ? 1.0 : 0.0
V = map(η,X,Y)
W = 0*V
```

Then the Strang splitting scheme is given by

```
V = TRBDF2_heat(V)
V = TRBDF2_heat(V')' |> Array
W = RK2(V,W,k/2)
V = TRBDF2(V,W,k)
W = RK2(V,W,k/2)
V = TRBDF2_heat(V)
V = TRBDF2_heat(V')' |> Array
```

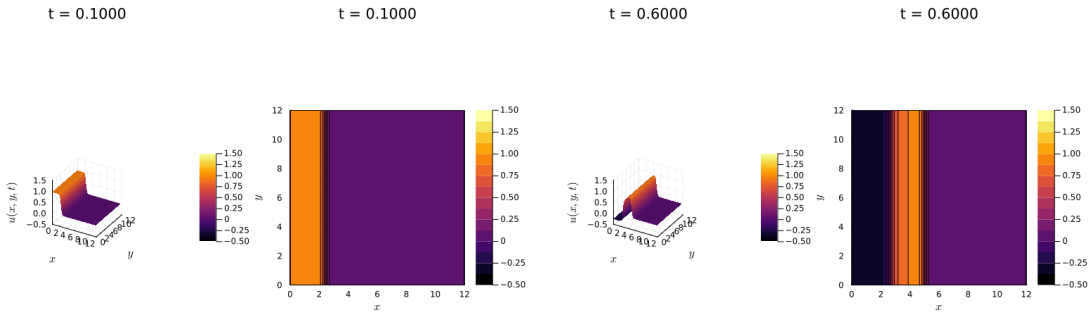
We set the parameters of the problem to be:

$$a = 0, \quad b = 12, \quad h = .05, \quad k = h/10, \quad \kappa = 1, \quad \epsilon = 0.01,$$
$$\alpha = 0.1, \quad \beta = 0.5, \quad \gamma = 1, \quad I_a = 0.$$

Now running our method, we see a single pulse travel across the domain in the positive x direction. We create a nice gif (see /src/code/6-1.gif) in the Notebook and here we show the solution $v(x,y,t)$ at times $t = 0.1, 0.6, 1.5,$ and 1.9. But now at $t = 0.9$, we zero out all the elements corresponding to y coordinates larger than 6, and we break the symmetry in the y-direction resulting in a spiral wave! We implement the interference with

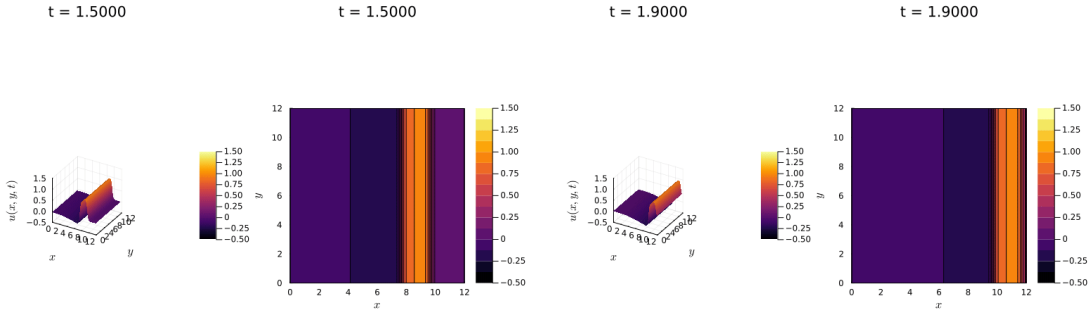
```
if t ≈ 0.9
    ind = floor((m + 2) / 2) |> Int
    V[1:ind, :] = zeros(ind, m + 2)
end
```

We create a nice gif in the Notebook (see /src/code/6-2.gif) but here we show the results at times $t = 0.9, 2, 3, 4, 5,$ and 6 which shows the spiraling behavior.



(a) Solution at $v(x, y, t)$ at $t = 0.1$.

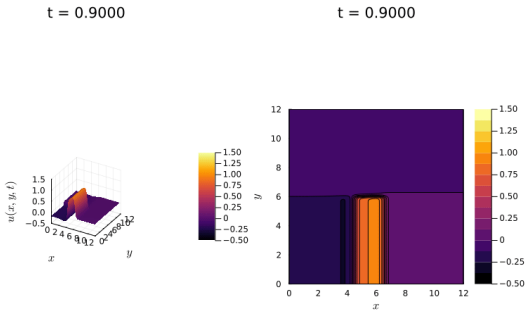
(b) Solution at $v(x, y, t)$ at $t = 0.6$.



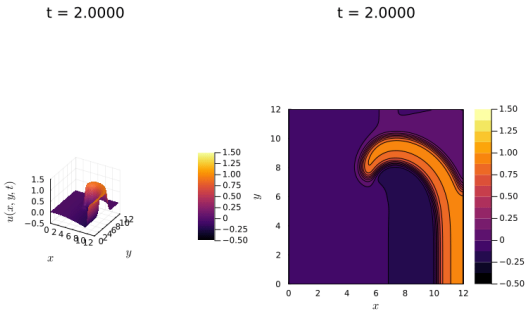
(c) Solution at $v(x, y, t)$ at $t = 1.5$.

(d) Solution at $v(x, y, t)$ at $t = 1.9$.

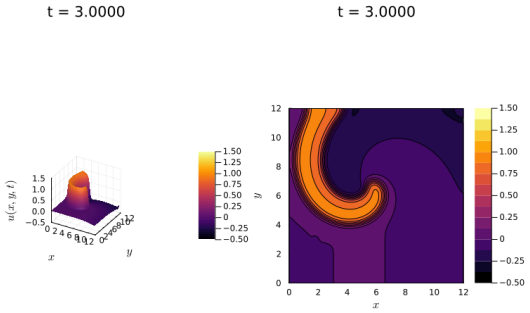
Figure 10: Solution $v(x, y, t)$ without interference at $t = 0.1, 0.6, 1.5,$ and 1.9.



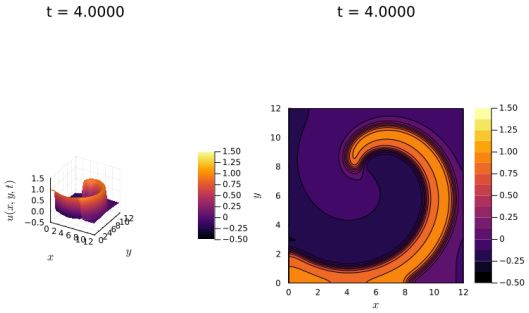
(a) Solution at $v(x, y, t)$ at $t = 0.9$.



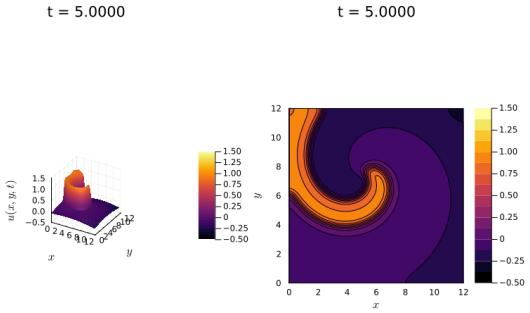
(b) Solution at $v(x, y, t)$ at $t = 2.0$.



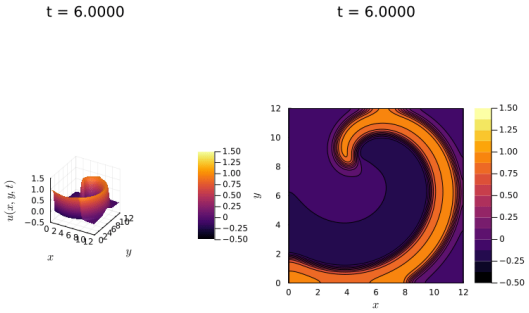
(c) Solution at $v(x, y, t)$ at $t = 3.0$.



(d) Solution at $v(x, y, t)$ at $t = 4.0$.



(e) Solution at $v(x, y, t)$ at $t = 5.0$.



(f) Solution at $v(x, y, t)$ at $t = 6.0$.

Figure 11: Solution $v(x, y, t)$ interference at $t = 0.9, 2, 3, 4, 5,$ and 6.

□