AMATH 586 Final

Cade Ballew #2120804

June 7, 2022

1 Problem 1

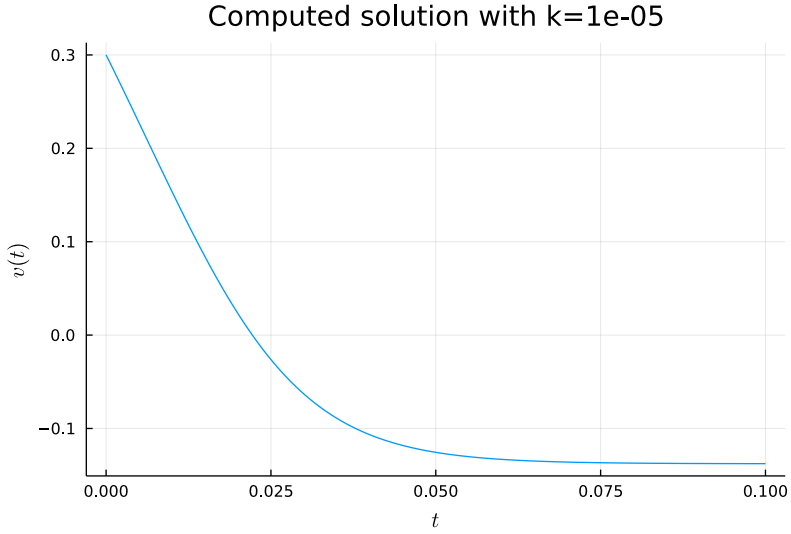
We consider the ODE

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

which we solve numerically using the TR-BDF-2 method,

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

in Julia. Using $v(0) = 0.3$, $\alpha = 0.5$, $w = 0.1$ and $\epsilon = 0.01$ as the parameters, we observe the following solution when solving to $t = 0.1$ with a stepsize of $k = 10^{-5}$.



To verify that our method is indeed second order accurate at $t = 0.1$, we treat the value that we found with this stepsize of $k = 10^{-5}$ as the true solution and compute the error and reduction ratio

$$\frac{\text{Error with time step } 2k}{\text{Error with time step } k}$$

for $k = 2^{-j}$ where $j = 4, \ldots, 10$. Doing this, our Julia code prints the following table.

```
    t       |       error        | reduction ratio
6.2500e-03 | 8.6012600408e-06 |
3.1250e-03 | 2.1200162577e-06 | 4.0571670191e+00
```

```
1.5625e-03 | 5.2458346345e-07 | 4.0413326105e+00
7.8125e-04 | 1.3036374619e-07 | 4.0239980730e+00
3.9063e-04 | 3.2473860656e-08 | 4.0144209390e+00
1.9531e-04 | 8.0895766474e-09 | 4.0142843157e+00
9.7656e-05 | 2.0048369631e-09 | 4.0350296789e+00
```

For an rth-order method we should see this be approximately 2^r, so this suggests that our method is indeed second order as most of our values are approximately 4.

2 Problem 2

Now, we consider the ODE system

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

by writing 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

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

and solve the problem numerically by Strange splitting, applying TR-BDF-2 to $\mathcal{A}$ and RK2 to $\mathcal{B}$. Taking

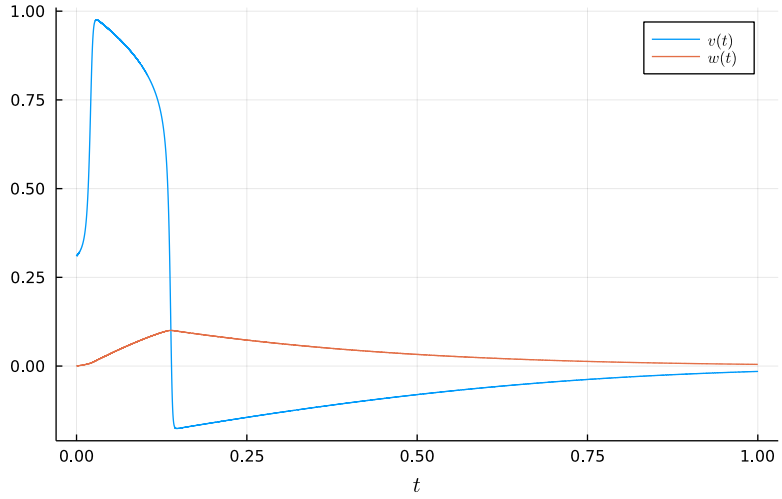
$$\alpha = 0.3, \quad \beta = 1, \quad I_a = 0, \quad \epsilon = 0.001,$$

with initial data

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

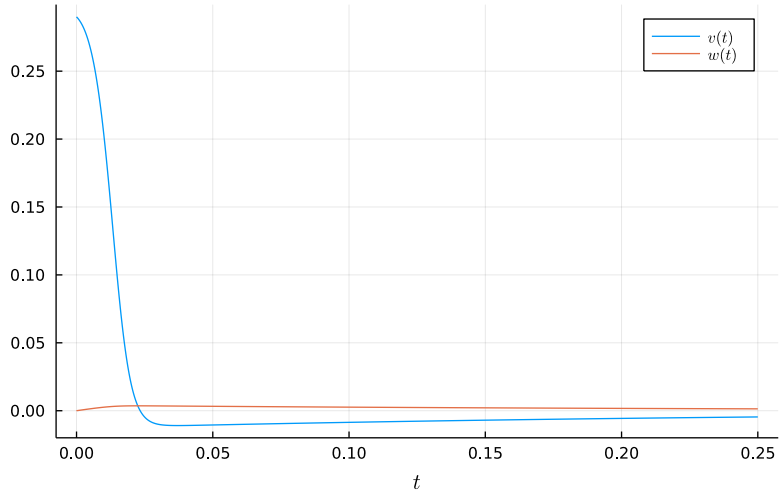
we observe the following when $v_0 = 0.31$ with small timestep $k = 10^{-5}$ and run until time $t = 1$.

Numerical solution with $v_0 = 0.31$



If we instead take $v_0 = 0.29$ and run to time $t = 0.25$, we observe the following.

Numerical solution with $v_0 = 0.29$



To verify that our method is indeed second order accurate at $t = 0.25$ with this choice of initial condition, we treat the value that we found with this stepsize of $k = 10^{-5}$ as the true solution and compute the error and reduction ratio

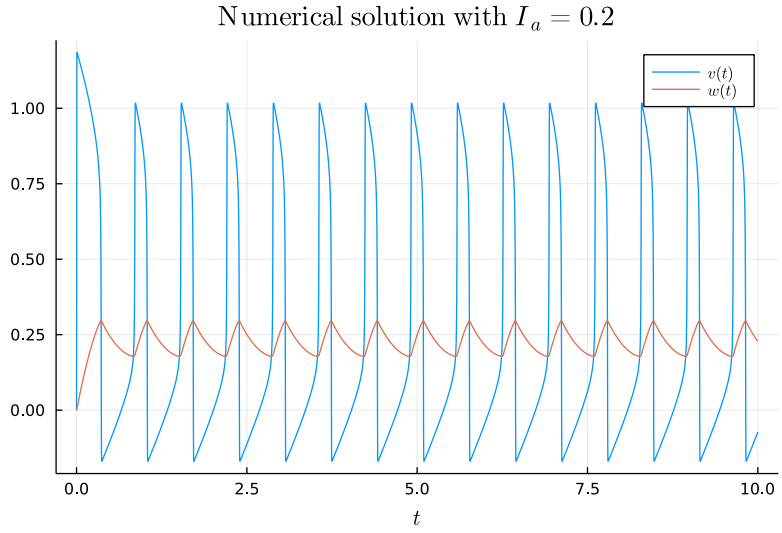
$$\frac{\text{Error with time step } 2k}{\text{Error with time step } k}$$

for $k = 2^{-j}$ where $j = 4, \ldots, 10$ for both v and w. Doing this, our Julia code prints the following table.

```
    t       |     error v      |     error w      |reduction ratio v |reduction ratio w
6.2500e-03 | 1.2097180179e-04 | 3.9611015529e-05 |
3.1250e-03 | 3.2088976949e-05 | 1.0536269722e-05 | 3.7698865246e+00 | 3.7594914115e+00
1.5625e-03 | 8.1769070508e-06 | 2.6897904284e-06 | 3.9243416551e+00 | 3.9171340676e+00
7.8125e-04 | 2.0594483574e-06 | 6.7813039540e-07 | 3.9704355885e+00 | 3.9664796721e+00
3.9063e-04 | 5.1631333772e-07 | 1.7009814519e-07 | 3.9887568399e+00 | 3.9867007053e+00
```

```
1.9531e-04 | 1.2902289893e-07 | 4.2517400749e-08 | 4.0017186253e+00 | 4.0006713062e+00
9.7656e-05 | 3.2025072342e-08 | 1.0554730014e-08 | 4.0288089765e+00 | 4.0282793301e+00
```

For an rth-order method we should see this be approximately 2^r, so this suggests that our method is indeed second order as most of our values are approximately 4 for both v and w.
Finally, we set $I_a = 0.2$ and $v_0 = 0$ and obtain the following plot.



3 Problem 3

Now, we consider the heat equation

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

with Neumann boundary conditions $v_x(a,t) = 0 = v_x(b,t)$ which has MOL discretization

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

where

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

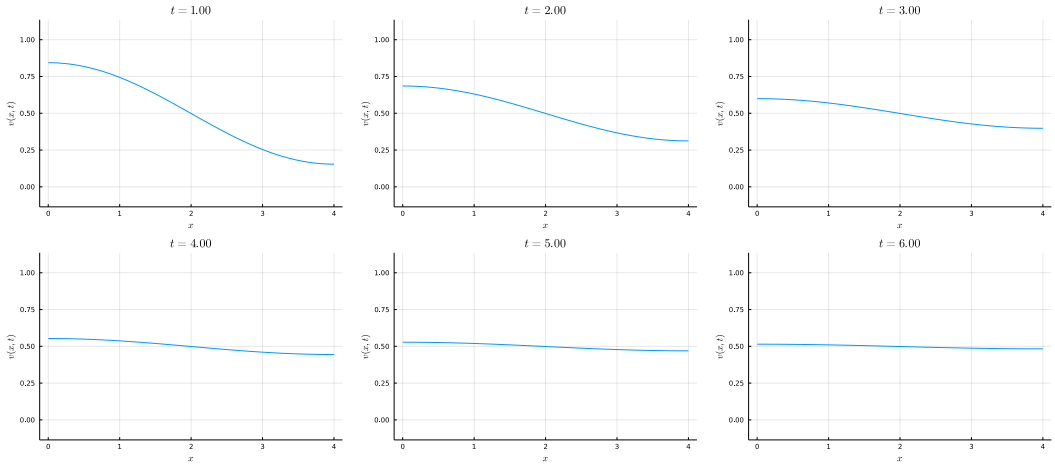
and

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

We apply TR-BDF-2 to this discretization where $a = 0$, $b = 4$, and initial data

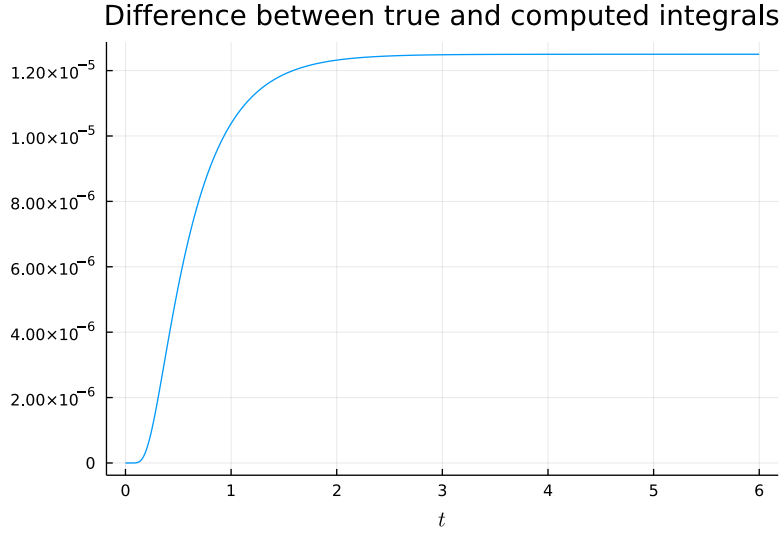
$$v(x,0) = \begin{cases} 1 & a \le x < \frac{a+b}{2} \\ 0 & \text{otherwise.} \end{cases}$$

Taking $h = 0.01$, $k = h$ and $\kappa = 1$, we solve this problem numerically to time $t = 6$. In the following, we display the computed solution at each integral time.



We also include the following gif of the computed solution over time.

While this solution appears to be converging to the long-time limit $v(x,t) \approx 1/2$ as $t \to \infty$, there is actually a small discrepancy between the numerical behavior and our expectation that the integral of the solution be nearly preserved. This would imply that the integral should be approximately 2 at each timestep, but if we plot the difference between the average value of our computed solution scaled by k[1] (so that it is an approximation to the integral) and this, we observe the following.



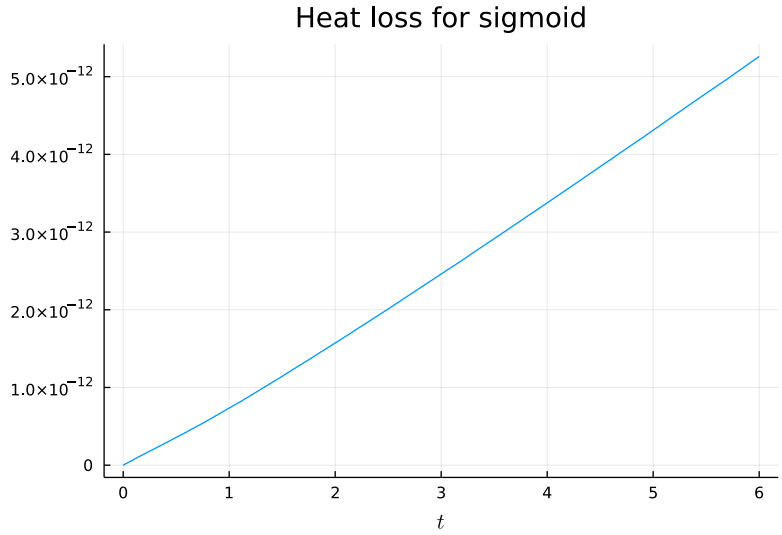
What we see here is that heat is actually being lost as our computed integral starts at the expected value of 2 but decreases as time goes on. The likely cause of this is the lack of smoothness in our initial condition. By trying different values of $h = k$, we observe that the heat loss appears to

[1] I.e., the 1 grid-norm minus 2

converge to a value that is $O(h^2)$[2] which suggests that this is due to the fact that our TR-BDF-2 method is second order accurate. Since the approximate integral that we are computing is just the L^1 grid-norm, we would expect second order accuracy, so this heat loss is consistent. However, the constant on our $O(h^2)$ term depends on the smoothness solution at each timestep, and since our initial condition is discontinuous, we expect a large error constant initially that decreases as our solution smooths; this is consistent with how our plot increases dramatically at first before smoothing out. If we instead use a shifted sigmoid function that is similar to $v(x,0)$ but smooth

$$1 - \frac{1}{1+e^{-10(x-2)}}$$

as our initial condition, we instead find a heat loss that is much closer to machine precision (pictured in the following plot) as the constant should be much smaller.



However, it is important to note that our sigmoid function does not satisfy our Neumann boundary conditions (but is quite close to doing do), so it is not quite proper to use this as an initial condition for our problem.

4 Problem 4

Now, we consider the ODE system

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

for $x \in (a,b)$. We take $I_a = 0$ and Strang split again into the heat equation term and our equation from problem 2. Doing this with the parameters

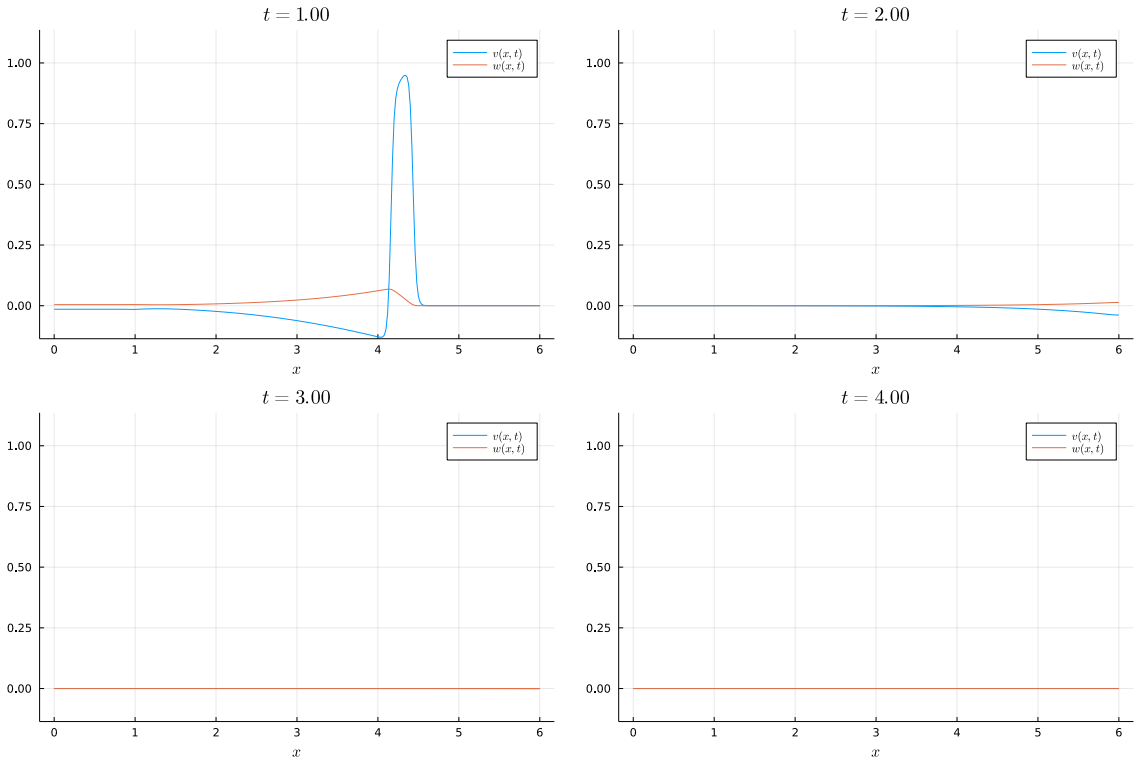
$$\begin{aligned}
h = 0.02, \quad k = h/10, \quad a = 0, \quad b = 6, \quad \alpha = 0.3,\\
\beta = 1, \quad \gamma = 1, \quad \kappa = 0.2, \quad \epsilon = 0.001.
\end{aligned}$$

[2] E.g. choosing $h = k = 0.1$ gives a heat loss of 1.2×10^{-3} instead.

and initial data

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

We solve the problem numerically to time $t = 4$ and observe the following computed solution at each integral time.



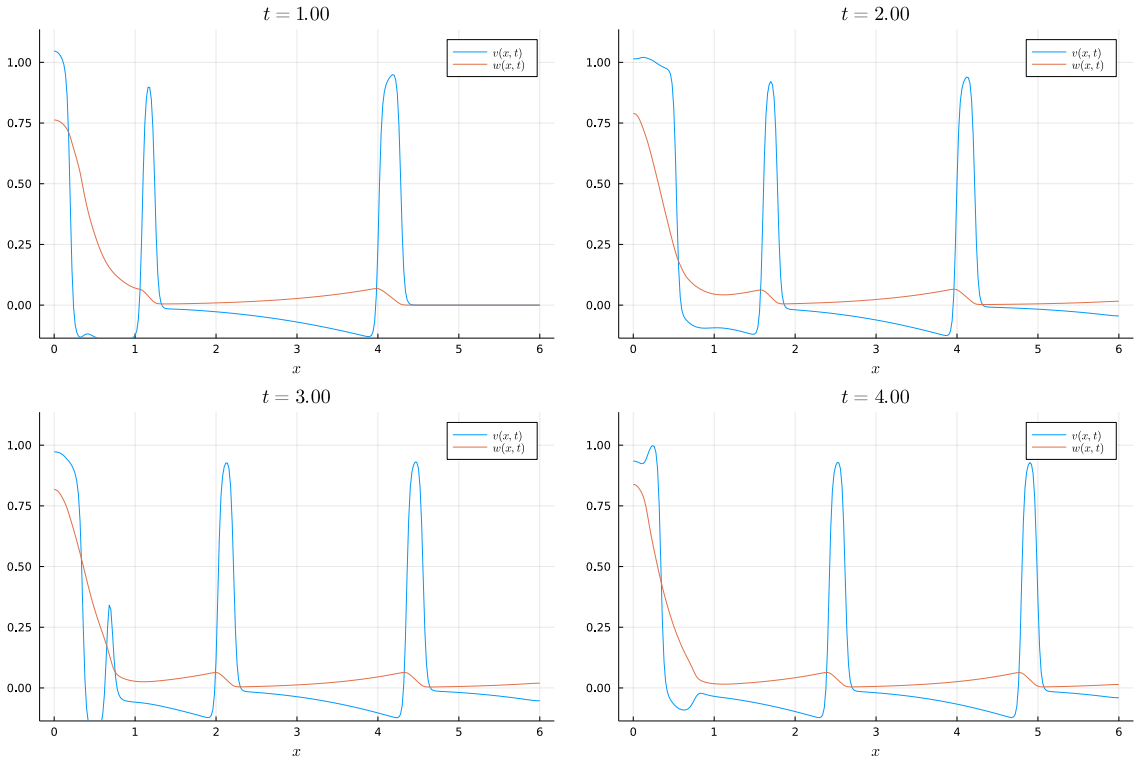
We also observe the following gif of the computed solution.

5 Problem 5

Now, we solve the same equation as in problem 4 but instead take

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

We solve the problem numerically to $t = 4$ and observe the following computed solution at each integral time.



We also observe the following gif of the computed solution.

6 Problem 6

Now, we consider the extension of problems 4 and 5 to two spatial dimensions: For $x \in [a,b],\ y \in [a,b]$:

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

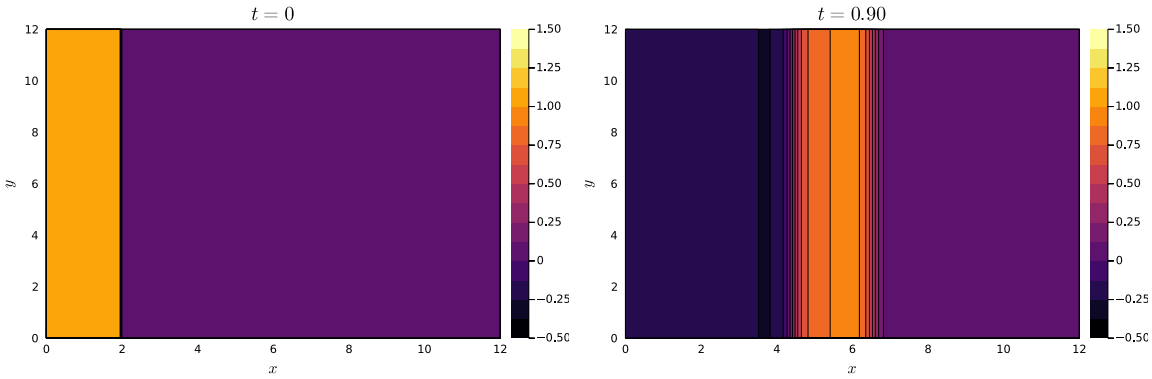
with initial and boundary data given by

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

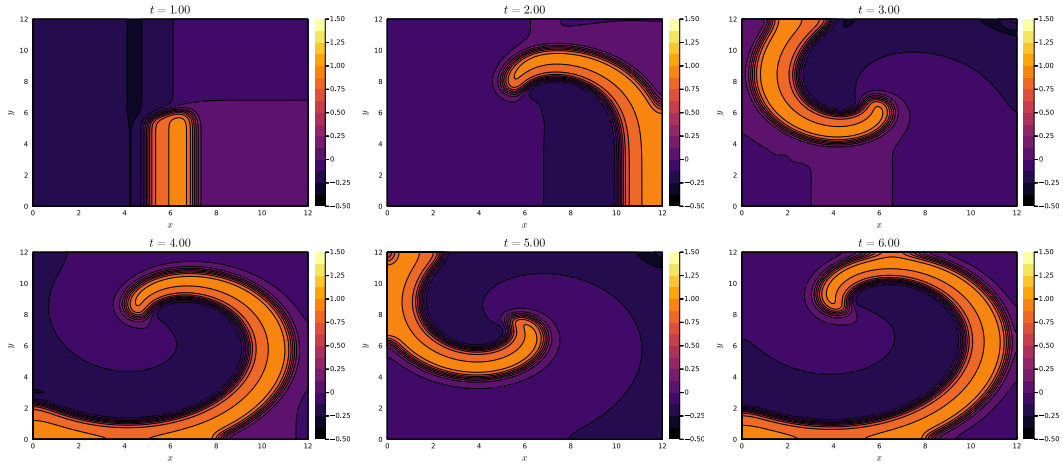
which we solve by Strang splitting in each spatial direction in addition to our prior method. Using parameters

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

and solving to time $t = 0.9$, our computed solution for v appears to be a single pulse that travels across the domain in the positive x-direction which we display at $t = 0$ and $t = 0.9$ in the following.



At time $t = 0.9$, we zero out all elements of V with associated y coordinates being larger than 6 to break the symmetry in the y-direction. Solving to time $t = 6$, we observe the following computed solution for v at each integral time.



We also observe the following gif of the computed solution.

7 Appendix A

The following Julia code is used for Problem 1.

```
using ForwardDiff, LinearAlgebra, Printf, Plots, LaTeXStrings

#Newton's method from demo
function Newton(x,g,Dg; tol = 1e-13, nmax = 1000)
    for j = 1:nmax
        step = Dg(x)\g(x)
        x -= step
        if maximum(abs.(step)) < tol
            break
        end
        if j == nmax
            println("Newton's method did not terminate")
        end
    end
    x
end
```

```
function TRBDF2(U,w,k)
    α, ϵ = 0.5, 0.01

    g(v) = v*(α-v)*(v-1)-w
    f(v) = g(v)/ϵ
    fprime(v) = ForwardDiff.derivative(f,v) #get derivative with autodiff

    G₁ = (u,Un) -> u - Un - (k/4)*(f(Un)+f(u))
    dG₁ = u -> 1-(k/4)*fprime(u) #derivative of G₁
    U₀ = Newton(U,u -> G₁(u,U), dG₁)

    G₂ = (u,Ustar,Un) -> u-(1/3)*(4Ustar-Un+k*f(u))
    dG₂ = u -> 1-(k/3)*fprime(u)

    Newton(U,u -> G₂(u,U₀,U), dG₂)
end

w = 0.1
T = 0.1
k = 1e-5 #chosen to be small
n = convert(Int64,ceil(T/k))
V = zeros(n+1)
V[1] = 0.3 #inital condition
for i = 2:n+1
    V[i] = TRBDF2(V[i-1], w, k)
end

p = plot(0:k:T,V, label=:false)
xlabel!(L"t")
ylabel!(L"v(t)")
title!(@sprintf("Computed solution with k=%0.0e",k))
savefig(p, "prob1.pdf")
display(p)


error = Inf
println("    t      |      error       | reduction ratio ")
for j = 4:10
    error_prev = error
    local k = 2.0^(-j)*0.1 #choose various values
    local n = convert(Int64,ceil(T/k))
    Vapprox = zeros(n+1)
    Vapprox[1] = 0.3 #inital condition
    for i = 2:n+1
        Vapprox[i] = TRBDF2(Vapprox[i-1], w, k)
    end
    global error = abs(V[end]-Vapprox[end])
    error_diff = error_prev/error
    println(@sprintf("%0.4e | %1.10e | %1.10e",k,error,error_diff))
end
```

The following Julia code is used for Problem 2.

```
using ForwardDiff, LinearAlgebra, Printf, Plots, LaTeXStrings

#Newton's method from demo
function Newton(x,g,Dg; tol = 1e-13, nmax = 1000)
    for j = 1:nmax
        step = Dg(x)\g(x)
        x -= step
        if maximum(abs.(step)) < tol
            break
        end
        if j == nmax
            println("Newton's method did not terminate")
        end
    end
    x
end

function TRBDF2(U,w,k)
    α, ϵ = 0.3, 0.001
```

```
    g(v) = v*(α-v)*(v-1)-w+I_a
    f(v) = g(v)/ϵ
    fprime(v) = ForwardDiff.derivative(f,v) #get derivative with autodiff

    G_1 = (u,Un) -> u - Un - (k/4)*(f(Un)+f(u))
    dG_1 = u -> 1-(k/4)*fprime(u)
    U_0 = Newton(U,u -> G_1(u,U), dG_1)

    G_2 = (u,Ustar,Un) -> u-(1/3)*(4Ustar-Un+k*f(u))
    dG_2 = u -> 1-(k/3)*fprime(u)

    Newton(U,u -> G_2(u,U_0,U), dG_2)
end

function RK2(V,W,k)
    β, γ = 1., 1.

    f(W,V) = β*V-γ*W
    W_0 = W + k*f(W,V)/2
    W += k*f(W_0,V)
end

T = 1.
k = 1e-5
n = convert(Int64,ceil(T/k))
global I_a = 0.
V = zeros(n+1)
W = zeros(n+1)
V[1] = 0.31 #inital condition
W[1] = 0.
for i = 2:n+1
    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)
end

p1 = plot(0:k:T, V, label=L"v(t)")
plot!(0:k:T, W, label=L"w(t)")
xlabel!(L"t")
title!(L"\mathrm{Numerical~solution~with}~v_0=0.31")
savefig(p1,"prob2a.pdf")
display(p1)

T = .25
n = convert(Int64,ceil(T/k))
V = zeros(n+1)
W = zeros(n+1)
V[1] = 0.29 #new inital condition
W[1] = 0.
for i = 2:n+1
    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)
end

p2 = plot(0:k:T, V, label=L"v(t)")
plot!(0:k:T, W, label=L"w(t)")
xlabel!(L"t")
title!(L"\mathrm{Numerical~solution~with}~v_0=0.29")
savefig(p2,"prob2b.pdf")
display(p2)

error1, error2 = Inf, Inf
println("    t      |     error v      |     error w
    |reduction ratio v |reduction ratio w")
for j = 4:10
    error1_prev = error1
    error2_prev = error2
    local k = 2.0^(-j)*0.1 #small
    local n = convert(Int64,ceil(T/k))
    Va = zeros(n+1)
    Wa = zeros(n+1)
    Va[1] = 0.29 #inital condition
    Wa[1] = 0.
    for i = 2:n+1
        Wa[i] = RK2(Va[i-1],Wa[i-1],k/2)
```

```
        Va[i] = TRBDF2(Va[i-1], Wa[i], k)
        Wa[i] = RK2(Va[i],Wa[i],k/2)
    end
    global error1 = abs(V[end]-Va[end])
    global error2 = abs(W[end]-Wa[end])
    error1_diff = error1_prev/error1
    error2_diff = error2_prev/error2
    println(@sprintf("%0.4e | %1.10e | %1.10e | %1.10e | %1.10e
        ",k,error1, error2, error1_diff, error2_diff))
end

T = 10.
k = 0.001
n = convert(Int64,ceil(T/k))
global I_a = 0.2 #make global to avoid passing into TRBDF2
V = zeros(n+1)
W = zeros(n+1)
V[1] = 0. #inital condition
W[1] = 0.
for i = 2:n+1
    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)
end

p3 = plot(0:k:T, V, label=L"v(t)")
plot!(0:k:T, W, label=L"w(t)")
xlabel!(L"t")
title!(L"\mathrm{Numerical~solution~with}~I_a=0.2")
savefig(p3,"prob2c.pdf")
display(p3)
```

The following Julia code is used for Problem 3.

```
using LinearAlgebra, Printf, Plots, LaTeXStrings


function TRBDF2_heat(V)
    V_0 = (I-(k/4)*B)\((I+(k/4)*B)*V)
    (I-(k/3)*B)\((1/3)*(4V_0-V))
end

h = 0.01
global k = h #global to avoid passing
a, b, κ = 0., 4., 1.
x = a:h:b
m = length(x)-2
T = 6.
n = convert(Int64,ceil(T/k))

A = Tridiagonal(fill(1.0,m+1),fill(-2.0,m+2),fill(1.0,m+1))
A[1,2] = 2.
A[end,end-1] = 2.
global B = (κ/h^2)*A #add scaling

V = zeros(m+2,n+1)
#implement initial condition
ind = x.<(a+b)/2
V[ind,1] .= 1

anim = Animation()
p = plot(x,V[:,1], yaxis = [-0.1,1.1], label=:false)
xlabel!(L"x")
ylabel!(L"v(x,t)")
title!(latexstring("t=0"))
frame(anim)
savefig("prob3_t=0.pdf")

for i=2:n+1
    V[:,i] = TRBDF2_heat(V[:,i-1])
    plot(x,V[:,i], yaxis = [-0.1,1.1], label=:false)
    if mod(i,2)==0
        xlabel!(L"x")
```

```
        ylabel!(L"v(x,t)")
        title!(latexstring(@sprintf("t=%1.2f",i*k)))
        frame(anim)
        if i*k ≈ round(i*k)
            savefig(@sprintf("prob3_t=%1.0f.pdf",i*k))
        end
    end
end
gif(anim,"prob3.gif")

avgV = h*sum(V,dims=1)
p = plot(0:k:T,-avgV'.+2, label=:false)
xlabel!(L"t")
title!("Difference between true and computed integrals")
savefig(p,"prob3_discrep.pdf")
display(p)

#try smoother initial condition
V2 = zeros(m+2,n+1)
sigmoid(x) = 1-1/(1+exp(-10(x-2)))
V2[:,1] = sigmoid.(x)
for i=2:n+1
    V2[:,i] = TRBDF2_heat(V2[:,i-1])
end

avgV2 = h*sum(V2,dims=1)
p1 = plot(0:k:T,abs.(avgV2'.-avgV2[1]), label=:false)
xlabel!(L"t")
title!("Heat loss for sigmoid")
savefig(p1,"prob3_sigmoid.pdf")
display(p1)
```

The following Julia code is used for Problem 4.

```
using ForwardDiff, LinearAlgebra, Printf, Plots, LaTeXStrings

#Newton's method from demo
function Newton(x,g,Dg; tol = 1e-8, nmax = 1000)
    for j = 1:nmax
        step = Dg(x)\g(x)
        x -= step
        if maximum(abs.(step)) < tol
            break
        end
        if j == nmax
            println("Newton's method did not terminate")
        end
    end
    x
end

#function created in problem 2
function TRBDF2_scalar(U,w,k)
    α, ϵ = 0.3, 0.001

    g(v) = v*(α-v)*(v-1)-w
    f(v) = g(v)/ϵ
    fprime(v) = ForwardDiff.derivative(f,v) #get derivative with autodiff

    G₁ = (u,Un) -> u - Un - (k/4)*(f(Un)+f(u))
    dG₁ = u -> 1-(k/4)*fprime(u)
    U₀ = Newton(U,u -> G₁(u,U), dG₁)

    G₂ = (u,Ustar,Un) -> u-(1/3)*(4Ustar-Un+k*f(u))
    dG₂ = u -> 1-(k/3)*fprime(u)

    Newton(U,u -> G₂(u,U₀,U), dG₂)
end

#use map to vectorize scalar TR-BDF-2 function
function TRBDF2(V,W,k)
    map((v,w) -> TRBDF2_scalar(v,w,k),V,W)
end
```

```
function RK2(V,W,k)
    β, γ = 1., 1.

    f(W,V) = β*V-γ*W
    W_0 = W + (k/2)*f(W,V)
    W += k*f(W_0,V)
end

function TRBDF2_heat(V)
    V_0 = (I-(k_2/4)*B)\((I+(k_2/4)*B)*V)
    (I-(k_2/3)*B)\((1/3)*(4V_0-V))
end

h = 0.02
k = h/10
global k_2 = k/2 #get half timestep
a, b, κ = 0., 6., 0.2
x = a:h:b
m = length(x)-2
T = 4.
n = convert(Int64,ceil(T/k))

A = Tridiagonal(fill(1.0,m+1),fill(-2.0,m+2),fill(1.0,m+1))
A[1,2] = 2.
A[end,end-1] = 2.
global B = (κ/h^2)*A #add scaling

V = zeros(m+2)
W = zeros(m+2)
#implement initial condition
ind = x.<1
V[ind] .= 1
anim = Animation()
plot(x,V, yaxis = [-0.1,1.1], label=L"v(x,t)")
plot!(x,W, yaxis = [-0.1,1.1], label=L"w(x,t)")
xlabel!(L"x")
title!(latexstring("t=0"))
frame(anim)
savefig("prob4_t=0.pdf")

for i=2:n+1
    global V = TRBDF2_heat(V)
    global W = RK2(V,W,k/2)
    V = TRBDF2(V,W,k)
    W = RK2(V,W,k/2)
    V = TRBDF2_heat(V)
    if mod(i,10)==0
        plot(x,V, yaxis = [-0.1,1.1], label=L"v(x,t)")
        plot!(x,W, yaxis = [-0.1,1.1], label=L"w(x,t)")
        xlabel!(L"x")
        title!(latexstring(@sprintf("t=%1.2f",i*k)))
        frame(anim)
        if mod(i,500)==0
            savefig(@sprintf("prob4_t=%1.0f.pdf",i*k))
        end
    end
end
gif(anim,"prob4.gif")
```

The following Julia code is used for Problem 5.

```
using ForwardDiff, LinearAlgebra, Printf, Plots, LaTeXStrings

#Newton's method from demo
function Newton(x,g,Dg; tol = 1e-8, nmax = 1000)
    for j = 1:nmax
        step = Dg(x)\g(x)
        x -= step
        if maximum(abs.(step)) < tol
            break
        end
        if j == nmax
```

```
            println("Newton's method did not terminate")
        end
    end
    x
end

#function created in problem 2
function TRBDF2_scalar(U,w,I_a,k)
    α, ϵ = 0.3, 0.001

    g(v) = v*(α-v)*(v-1)-w+I_a
    f(v) = g(v)/ϵ
    fprime(v) = ForwardDiff.derivative(f,v) #get derivative with autodiff

    G_1 = (u,Un) -> u - Un - (k/4)*(f(Un)+f(u))
    dG_1 = u -> 1-(k/4)*fprime(u)
    U_0 = Newton(U,u -> G_1(u,U), dG_1)

    G_2 = (u,Ustar,Un) -> u-(1/3)*(4Ustar-Un+k*f(u))
    dG_2 = u -> 1-(k/3)*fprime(u)

    Newton(U,u -> G_2(u,U_0,U), dG_2)
end

#use map to vectorize scalar TR-BDF-2 function
function TRBDF2(V,W,k)
    map((v,w,i_a) -> TRBDF2_scalar(v,w,i_a,k),V,W,I_avec)
end

function RK2(V,W,k)
    β, γ = 1., 1.

    f(W,V) = β*V-γ*W
    W_0 = W + (k/2)*f(W,V)
    W += k*f(W_0,V)
end

function TRBDF2_heat(V)
    V_0 = (I-(k_2/4)*B)\((I+(k_2/4)*B)*V)
    (I-(k_2/3)*B)\((1/3)*(4V_0-V))
end

h = 0.02
k = h/10
global k_2 = k/2 #get half timestep
a, b, κ = 0., 6., 0.2
x = a:h:b
m = length(x)-2
T = 4.
n = convert(Int64,ceil(T/k))

A = Tridiagonal(fill(1.0,m+1),fill(-2.0,m+2),fill(1.0,m+1))
A[1,2] = 2.
A[end,end-1] = 2.
global B = (κ/h^2)*A #add scaling

V = zeros(m+2)
W = zeros(m+2)

anim = Animation()
plot(x,V, yaxis = [-0.1,1.1], label=L"v(x,t)")
plot!(x,W, yaxis = [-0.1,1.1], label=L"w(x,t)")
xlabel!(L"x")
title!(latexstring("t=0"))
frame(anim)
savefig("prob5_t=0.pdf")

global I_avec = @. 0.8exp(-5x^2)

for i=2:n+1
    global V = TRBDF2_heat(V)
    global W = RK2(V,W,k/2)
    V = TRBDF2(V,W,k)
    W = RK2(V,W,k/2)
    V = TRBDF2_heat(V)
    if mod(i,10)==0
```

```
        plot(x,V, yaxis = [-0.1,1.1], label=L"v(x,t)")
        plot!(x,W, yaxis = [-0.1,1.1], label=L"w(x,t)")
        xlabel!(L"x")
        title!(latexstring(@sprintf("t=%1.2f",i*k)))
        frame(anim)
        if mod(i,500)==0
            savefig(@sprintf("prob5_t=%1.0f.pdf",i*k))
        end
    end
end
gif(anim,"prob5.gif")
```

The following Julia code is used for Problem 6.

```
using ForwardDiff, LinearAlgebra, Printf, Plots, LaTeXStrings

#Newton's method from demo
function Newton(x,g,Dg; tol = 1e-8, nmax = 1000)
    for j = 1:nmax
        step = Dg(x)\g(x)
        x -= step
        if maximum(abs.(step)) < tol
            break
        end
        if j == nmax
            println("Newton's method did not terminate")
        end
    end
    x
end

#function created in problem 2
function TRBDF2_scalar(U,w,k)
    α, ϵ = 0.1, 0.01

    g(v) = v*(α-v)*(v-1)-w
    f(v) = g(v)/ϵ
    fprime(v) = ForwardDiff.derivative(f,v) #get derivative with autodiff

    G₁ = (u,Un) -> u - Un - (k/4)*(f(Un)+f(u))
    dG₁ = u -> 1-(k/4)*fprime(u)
    U₀ = Newton(U,u -> G₁(u,U), dG₁)

    G₂ = (u,Ustar,Un) -> u-(1/3)*(4Ustar-Un+k*f(u))
    dG₂ = u -> 1-(k/3)*fprime(u)

    Newton(U,u -> G₂(u,U₀,U), dG₂)
end

#use map to vectorize scalar TR-BDF-2 function
function TRBDF2(V,W,k)
    map((v,w) -> TRBDF2_scalar(v,w,k),V,W)
end

function RK2(V,W,k)
    β, γ = 0.5, 1.

    f(W,V) = β*V-γ*W
    W₀ = W + (k/2)*f(W,V)
    W += k*f(W₀,V)
end

function TRBDF2_heat(V)
    V₀ = (I-(k₂/4)*B)\((I+(k₂/4)*B)*V)
    (I-(k₂/3)*B)\((1/3)*(4V₀-V))
end

h = 0.05
k = h/10
global k₂ = k/2 #get half timestep
a, b, κ = 0., 12., 1.
T = 0.9
n = convert(Int64,ceil(T/k))
```

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

m = length(x)-2
A = Tridiagonal(fill(1.0,m+1),fill(-2.0,m+2),fill(1.0,m+1))
A[1,2] = 2.
A[end,end-1] = 2.
global B = (κ/h^2)*A #add scaling

anim = Animation()
contourf(x,y,reverse(V,dims=1), clim=(-.5,1.5))
xlabel!(L"x")
ylabel!(L"y")
title!(latexstring("t=0"))
frame(anim)
savefig("prob6_t=0.pdf")

for i = 2:n+1
    global V = TRBDF2_heat(V)
    V = TRBDF2_heat(V')' |> Array
    global 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
    if mod(i,5)==0
        contourf(x,y,reverse(V,dims=1),clim=(-.5,1.5))
        xlabel!(L"x")
        ylabel!(L"y")
        title!(latexstring(@sprintf("t=%1.2f",i*k)))
        frame(anim)
        if mod(i,500)==0
            savefig(@sprintf("prob6_t=%1.0f.pdf",i*k))
        end
    end
end
savefig("prob6_t=0.9.pdf")

Dn = (x,y) -> y <= 6
V = map(Dn,X,Y).*V #zero out necessary entries
T = 6.0-0.9
n = convert(Int64,ceil(T/k))
for i = 2:n+1
    global V = TRBDF2_heat(V)
    V = TRBDF2_heat(V')' |> Array
    global 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
    if mod(i,5)==0
        #need to flip matrix when plotting to account for ordering convention
        contourf(x,y,reverse(V,dims=1),clim=(-.5,1.5))
        xlabel!(L"x")
        ylabel!(L"y")
        title!(latexstring(@sprintf("t=%1.2f",i*k+0.9)))
        frame(anim)
        if i*k+0.9 ≈ round(i*k+0.9)
            savefig(@sprintf("prob6_t=%1.0f.pdf",i*k+0.9))
        end
    end
end

gif(anim,"prob6.gif")
```