Progsheet 1 - Introduction to the Numerics of partial differential equations

Author: Sven Ullmann and Jonathan Schnitzler

This is an exercise sheet in one space and one time dimension.

Cars on a road

We consider the initial-boundary value-problem for a unknown $u \in C^1(\Omega_T)$

$$\partial_t u(x,t) + \partial_x F[u(x,t)] = 0 \qquad \text{in } \Omega_T$$

$$u(\cdot, 0) = u_0 \qquad \text{in } \Omega$$

$$\partial_x u(x_{\min}, \cdot) = \partial_x u(x_{\max}, \cdot) \qquad \text{in } (0, T)$$

We define

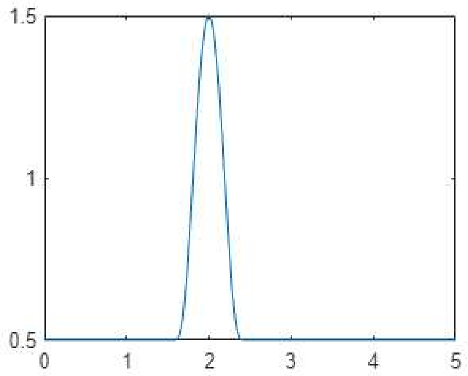
```
%Domain
x_min = 0;
x_max = 5;
T = 1;


%Discretication
nt = 100;
dt = T/nt,
```

```
dt = 0.0100
```

```

nx = 250;
dx = (x_max - x_min)/nx,
```

```
dx = 0.0200
```

```


omega = linspace(x_min, x_max, nx);
omega_time = linspace(0,T,nt);


%Boundary values
u0 = initial_values(omega);
plot(omega, u0)
```




```

%Non Linear Function F
f = @(u) u.*(1-u)
```

```
f = function_handle with value:
    @(u)u.*(1-u)
```

Flux kernels G

We will consider different flux kernels with which we will solve the iterative scheme for this PDE

General iterative scheme

$$u_j^{(n+1)} := u_j^n - \frac{\Delta t}{\Delta x}\left(G^n_{\left(j+\frac{1}{2}\right)} - G^n_{\left(j-\frac{1}{2}\right)}\right)$$

$u_J^n := u_0^n$ (cars are in a closed loop)

```
%Initialization
u = zeros(nt,nx);
u(1,:) = u0;

%Placeholder for flux kernel
G = @(u,n,j,dt,dx) 0            %+1/2 -> 1 and -1/2 ->0
```

```
G = function_handle with value:
    @(u,n,j,dt,dx)0
```

```
% SPECIAL CASE: j = 0 and flux 0-1/2
Gs = @(u,n,dt,dx) 0
```

```
Gs = function_handle with value:
    @(u,n,dt,dx)0
```

```
for n = 1:(nt-1)
        u(n+1, 1) = u(n,1) - dt/dx .* (G(u,n,2,dt,dx) - Gs(u,n,dt,dx));
        u(n+1, end) = u(n+1,1);
        for j = 2:(nx-1)
            u(n+1, j) = u(n,j) - dt/dx .* (G(u,n,j+1,dt,dx)- G(u,n,j,dt,dx));
        end
end

%Which can be vectorized by assembling to a matrix
```

In the following we collect the implementation of the different fluxes, which are also saved as a `flux.m` file where the specification is entered as the last argument with a string

`spec = lax` or

`spec = up` or lastly

`spec = down`.

Lax-Friedrichs

$$G^n_{j+\frac{1}{2}} = \frac{u_j^n\left(1-u_j^n\right) + u_{j+1}^n\left(1-u_{j+1}^n\right)}{2} - \frac{\Delta x}{2\Delta t}\left(u_{j+1}^n - u_j^n\right)$$

$$G^n_{-\frac{1}{2}} = \frac{u_J^n\left(1-u_J^n\right) + u_1^n\left(1-u_1^n\right)}{2} - \frac{\Delta x}{2\Delta t}\left(u_1^n - u_J^n\right)$$

```
G = @(u,n,j,dt,dx) (u(n,j-1).*(1-u(n,j-1)) + u(n,j).*(1-u(n,j)))/2 - dx/(2.*dt).*(u(n,j)-u(n,j-1));
Gs = @(u,n,dt,dx) (u(n,end).*(1-u(n,end)) + u(n,1).*(1-u(n,1)))/2 - dx/(2.*dt).*(u(n,1)-u(n,end));
% Assuming there is a typo in the worksheet, and it should use the first
% value (i.e. u_0)
```

Anmerkung: In python könnte man Gs sehr elegant implementieren, da bei negativer indiziierung der von dem letzten Wert gezählt wird, was genau die richtige Funktion implementieren würde.

"Upwind"-Flux

$$G^n_{j+\frac{1}{2}} = u_j^n\left(1-u_j^n\right)$$

$$G^n_{-\frac{1}{2}} = u_J^n\left(1-u_J^n\right)$$

```
G = @(u,n,j,dt,dx) u(n,j-1).*(1-u(n,j-1));
Gs = @(u,n,dt,dx) u(n,end).*(1-u(n,end));
```

"Downwind"-Flux

$$G^n_{j+\frac{1}{2}} = u_{j+1}^n\left(1-u_{j+1}^n\right)$$

$$G^n_{-\frac{1}{2}} = u_0^n\left(1-u_0^n\right)$$

```
G = @(u,n,j,dt,dx) u(n,j).*(1-u(n,j));
```

```
Gs = @(u,n,dt,dx) u(n,1).*(1-u(n,1));
```

I used different notation since I found +1/2 is a bit awkward i just fancied clearity over symmetrie and choose $j+\frac{1}{2} \to j+1$ and $j-\frac{1}{2} \to j$

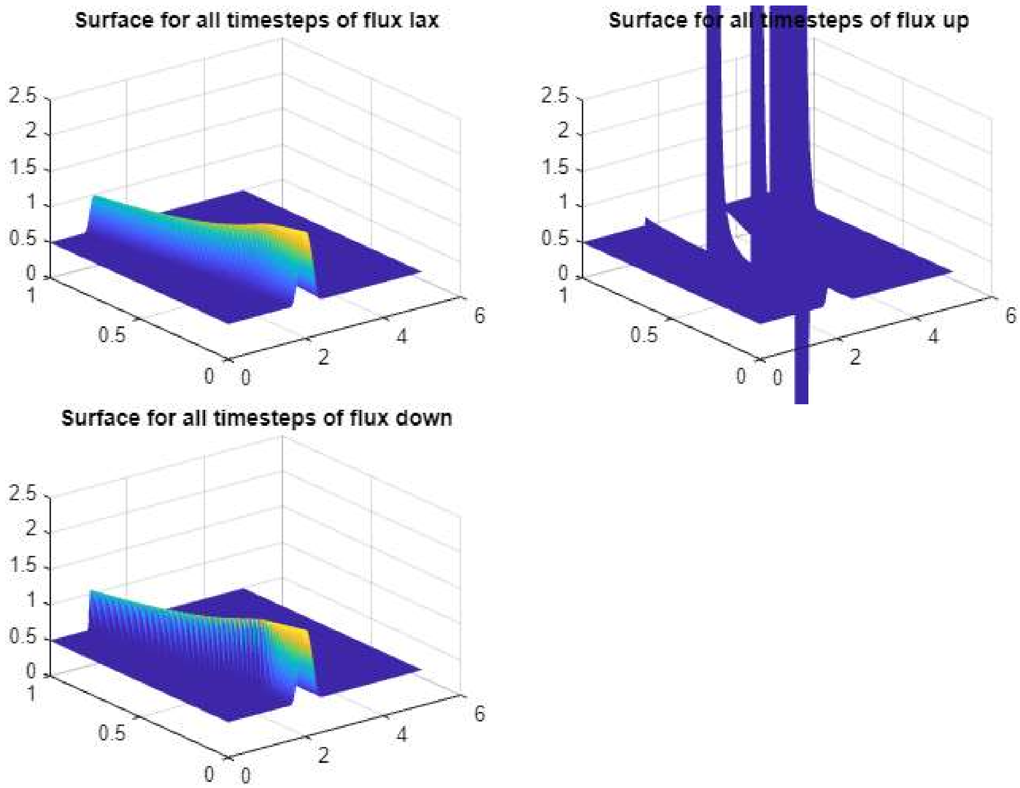
Task b)

Already in general form used in c)

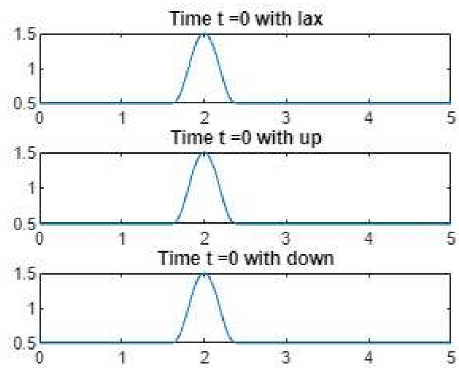
```
type = ["lax","up","down"];
timeInstances = 0:0.5:T, %Generate multiple evaluation for time instances
```

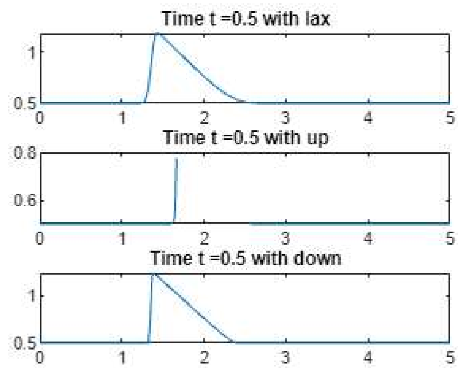
```
timeInstances = 1×3
         0    0.5000    1.0000
```

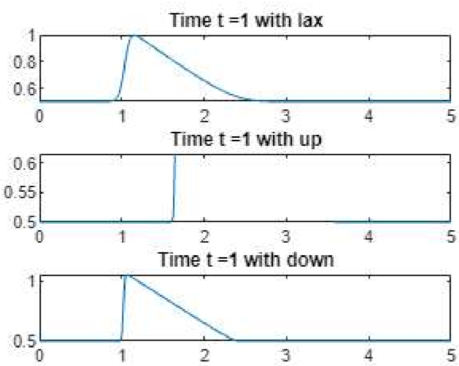
```



omega = linspace(x_min, x_max, nx);
omega_time = linspace(0,T,nt);

u0 = initial_values(omega);
uAll = zeros(nt,nx,numel(type));

%Calculation for all three types
for ty = 1:numel(type)
    u = zeros(nt,nx);
    u(1,:) = u0;
    [G,Gs] = flux(type(ty));


    for n = 1:(nt-1)
        u(n+1, 1) = u(n,1) - dt/dx .* (G(u,n,2,dt,dx) - Gs(u,n,dt,dx));
        u(n+1, end) = u(n+1,1);
        for j = 2:(nx-1)
            u(n+1, j) = u(n,j) - dt/dx .* (G(u,n,j+1,dt,dx)- G(u,n,j,dt,dx));
        end
    end

    %Surface plots
    uAll(:,:,ty) = u;
    [X, Y] = meshgrid(omega,omega_time);
    figure
    s = surf(X,Y,u);
    s.EdgeColor = 'none';
    zlim([0 2.5])
    title("Surface for all timesteps of flux " + type(ty))


end
```




```
%Plotting


figure
for ti = 1:numel(timeInstances)
    figure
    for ty = 1:numel(type)
        subplot(numel(type),1,ty)
%             index of timeInstance -> floor(ti./T.*(nt-1))+1
        plot(omega,uAll(floor(timeInstances(ti)./T.*(nt-1))+1,:,ty))
        title("Time t =" + string(timeInstances(ti)) + " with " + type(ty))
    end
end
```







In our programm the qualitative difference of the numerical fluxes is that

- **upwind** flux seems to be divergent and unstable
- **downwind** flux and **lax-friedrichs** are both transporting the cars or concentration to the left towards the 0 coordinate
- **downwind** has rough edges
- **lax-friedrichs** seems more smooth

Task c)

We wrote this programm in a real general form, but maybe it is even better to split at least the last task b) and c) in two cells to have some place to talk about the programm.

Maintaining of the spatial grid

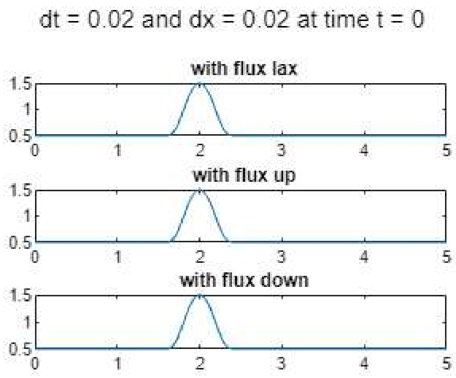
```
type = ["lax","up","down"];
timeInstances = 0:0.5:T, %Generate multiple evaluation for time instances
```

```
timeInstances = 1×3
         0    0.5000    1.0000
```

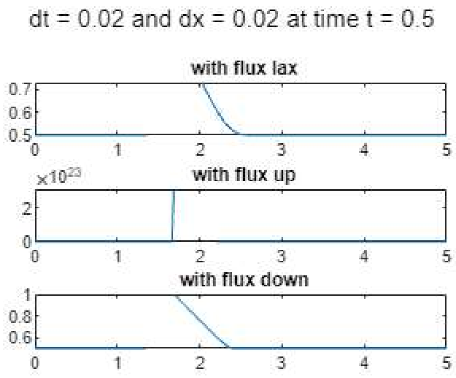
```

ntArr = [50,100,200];
dtArr = T./ntArr;
nx = 250;

for nt = ntArr
    dt = T/nt,
    dx = (x_max-x_min)./nx



    omega = linspace(x_min, x_max, nx);
    omega_time = linspace(0,T,nt);

    u0 = initial_values(omega);
    uAll = zeros(nt,nx,numel(type));

    %Calculation for all three types
    for ty = 1:numel(type)
        u = zeros(nt,nx);
        u(1,:) = u0;
        [G,Gs] = flux(type(ty));


        for n = 1:(nt-1)
            u(n+1, 1) = u(n,1) - dt/dx .* (G(u,n,2,dt,dx) - Gs(u,n,dt,dx));
            u(n+1, end) = u(n+1,1);
            for j = 2:(nx-1)
                u(n+1, j) = u(n,j) - dt/dx .* (G(u,n,j+1,dt,dx)- G(u,n,j,dt,dx));
            end
        end

%             %Surface plots
            uAll(:,:,ty) = u;
%             [X, Y] = meshgrid(omega,omega_time);
%             figure
%             surf(X,Y,u)
%             zlim([0 2.5])


    end


    %Plotting

    figure
    for ti = 1:numel(timeInstances)
        idx_t = floor(timeInstances(ti)./T.*(nt-1))+1;
        figure
```

```
        for ty = 1:numel(type)
            subplot(numel(type),1,ty)
            plot(omega,uAll(idx_t,:,ty))
            title(" with flux " + type(ty))
        end
        sgtitle("dt = " + string(dt) + " and dx = " + string(dx) + " at time t = " + string(timeInstances(ti)))

        %Sum over domain
        sumOverDomain = sum(uAll(idx_t, :,:),2)
    end


end
```

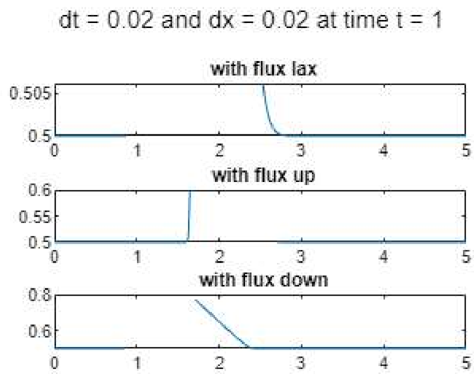
```
dt = 0.0200
dx = 0.0200
```



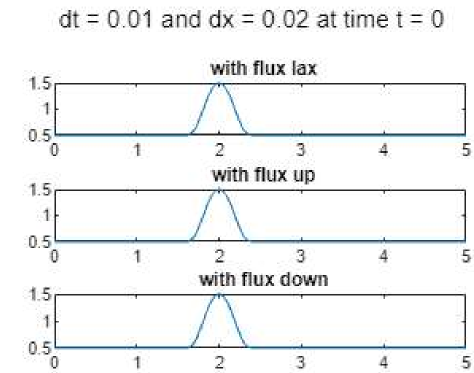

```
sumOverDomain = 
sumOverDomain(:,:,1) =

  144.1141


sumOverDomain(:,:,2) =

  144.1141


sumOverDomain(:,:,3) =

  144.1141
```



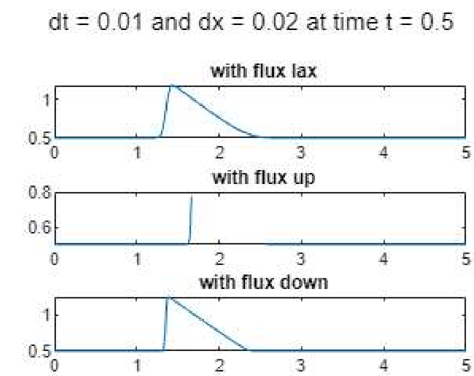

```
sumOverDomain = 
sumOverDomain(:,:,1) =

   NaN


sumOverDomain(:,:,2) =

   NaN


sumOverDomain(:,:,3) =

   NaN
```




```
sumOverDomain =
sumOverDomain(:,:,1) =

   NaN


sumOverDomain(:,:,2) =

   NaN


sumOverDomain(:,:,3) =

   NaN
dt = 0.0100
dx = 0.0200
```

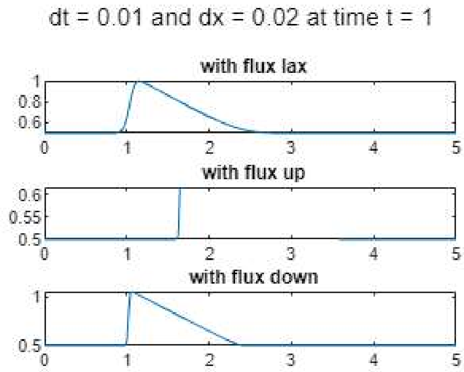



```
sumOverDomain =
sumOverDomain(:,:,1) =

  144.1141


sumOverDomain(:,:,2) =

  144.1141


sumOverDomain(:,:,3) =

  144.1141
```

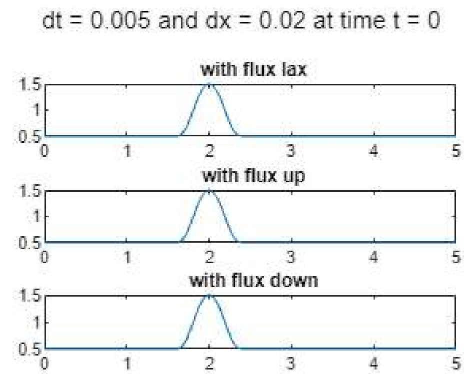



```
sumOverDomain =
```

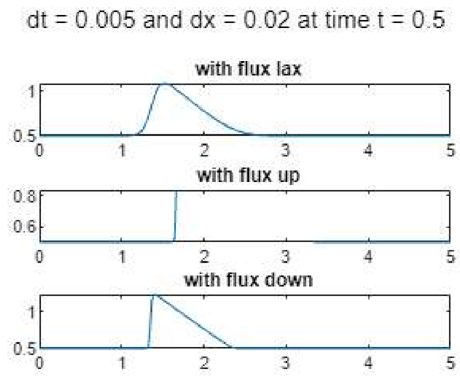
```
sumOverDomain(:,:,1) =

  144.1141


sumOverDomain(:,:,2) =

   NaN


sumOverDomain(:,:,3) =

  144.1141
```



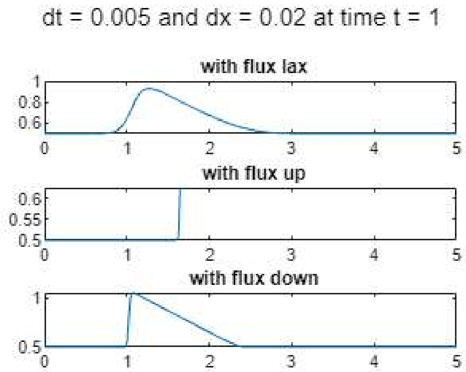

```
sumOverDomain =
sumOverDomain(:,:,1) =

  144.1141


sumOverDomain(:,:,2) =

   NaN


sumOverDomain(:,:,3) =

  144.1141
dt = 0.0050
dx = 0.0200
```




```
sumOverDomain =
sumOverDomain(:,:,1) =

  144.1141


sumOverDomain(:,:,2) =

  144.1141


sumOverDomain(:,:,3) =

  144.1141
```




```
sumOverDomain =
sumOverDomain(:,:,1) =

  144.1141


sumOverDomain(:,:,2) =

   NaN


sumOverDomain(:,:,3) =

  144.1141
```




```
sumOverDomain =
sumOverDomain(:,:,1) =

  144.1141


sumOverDomain(:,:,2) =

   NaN


sumOverDomain(:,:,3) =

  144.1141
```

One can see for

- Lower refinement of time-steps $\mathrm{dt} = 0.02$ result in divergent behaviour for all methods
- $\mathrm{dt} = 0.01$ is already described in b)
- $\mathrm{dt} = 0.005$ lax is more smooth without sharp edge

Task c)

Simultaneousl shrinking the spatial and temporal step with identical factor

```
type = ["lax","up","down"];
timeInstances = 0:0.5:T, %Generate multiple evaluation for time instances
```

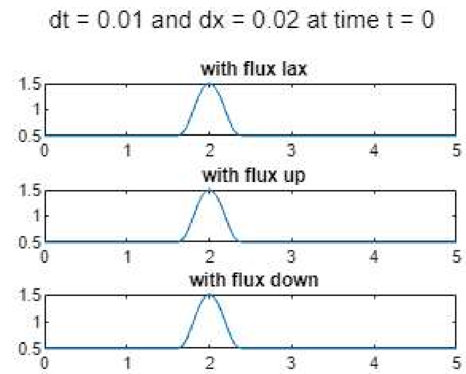
```
timeInstances = 1×3
         0    0.5000    1.0000
```

```
ntBase = 100;
nxBase = 250;
refineFactor = [1, 2, 4]
```

```
refineFactor = 1×3
     1     2     4
```

```
for fac = refineFactor
    nt = ntBase.*fac;
    dt = T/nt,

    nx = nxBase.*fac
    dx = (x_max-x_min)./nx



    omega = linspace(x_min, x_max, nx);
    omega_time = linspace(0,T,nt);

    u0 = initial_values(omega);
    uAll = zeros(nt,nx,numel(type));

    %Calculation for all three types
    for ty = 1:numel(type)
        u = zeros(nt,nx);
        u(1,:) = u0;
        [G,Gs] = flux(type(ty));


        for n = 1:(nt-1)
            u(n+1, 1) = u(n,1) - dt/dx .* (G(u,n,2,dt,dx) - Gs(u,n,dt,dx));
            u(n+1, end) = u(n+1,1);
            for j = 2:(nx-1)
                u(n+1, j) = u(n,j) - dt/dx .* (G(u,n,j+1,dt,dx)- G(u,n,j,dt,dx));
            end
        end

        uAll(:,:,ty) = u;
%             %Surface plots
%             [X, Y] = meshgrid(omega,omega_time);
%             figure
%             surf(X,Y,u)
%             zlim([0 2.5])


    end


    %Plotting


    figure
    for ti = 1:numel(timeInstances)
        idx_t = floor(timeInstances(ti)./T.*(nt-1))+1;
        figure

        for ty = 1:numel(type)
            subplot(numel(type),1,ty)
            plot(omega,uAll(idx_t,:,ty))
           title(" with flux " + type(ty))
        end
        sgtitle("dt = " + string(dt) + " and dx = " + string(dx) + " at time t = " + string(timeInstances(ti)))

        %Sum over domain
        sumOverDomain = sum(uAll(idx_t, :,:),2)
    end

end
```

```
dt = 0.0100
nx = 250
dx = 0.0200
```



sumOverDomain =

sumOverDomain(:,:,1) =

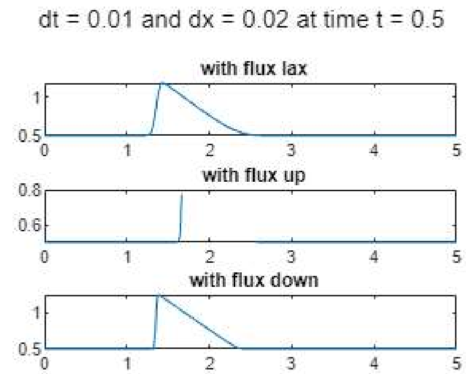
    144.1141

sumOverDomain(:,:,2) =

    144.1141

sumOverDomain(:,:,3) =

    144.1141



sumOverDomain =

sumOverDomain(:,:,1) =

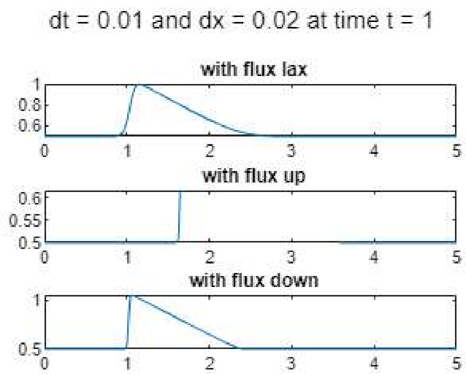
    144.1141

sumOverDomain(:,:,2) =

    NaN

sumOverDomain(:,:,3) =

    144.1141

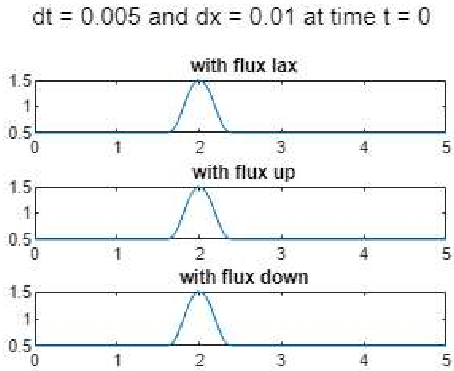


sumOverDomain =

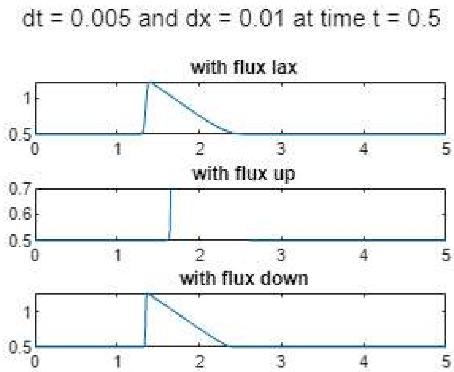
sumOverDomain(:,:,1) =

    144.1141

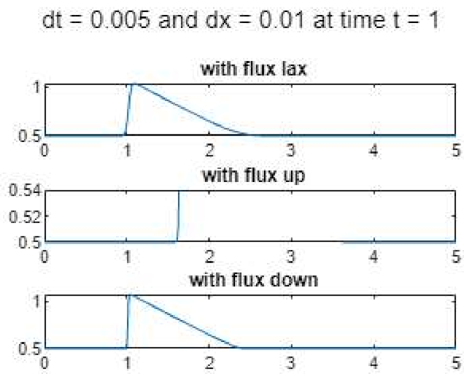
```
sumOverDomain(:,:,2) =

   NaN


sumOverDomain(:,:,3) =

  144.1141
dt = 0.0050
nx = 500
dx = 0.0100
```



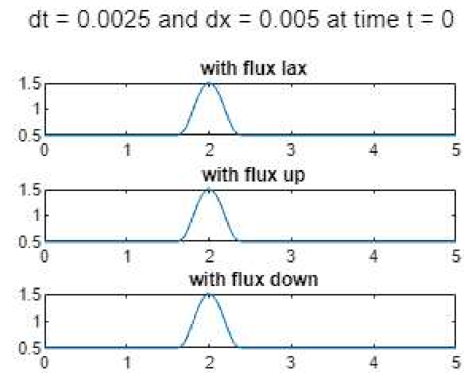

```
sumOverDomain =
sumOverDomain(:,:,1) =

  288.3050


sumOverDomain(:,:,2) =

  288.3050


sumOverDomain(:,:,3) =

  288.3050
```



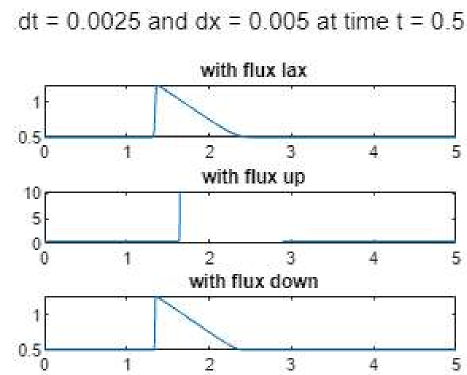

```
sumOverDomain =
sumOverDomain(:,:,1) =

  288.3050


sumOverDomain(:,:,2) =

   NaN


sumOverDomain(:,:,3) =

  288.3050
```




```
sumOverDomain =
sumOverDomain(:,:,1) =

  288.3050


sumOverDomain(:,:,2) =

   NaN


sumOverDomain(:,:,3) =

  288.3050
dt = 0.0025
nx = 1000
dx = 0.0050
```

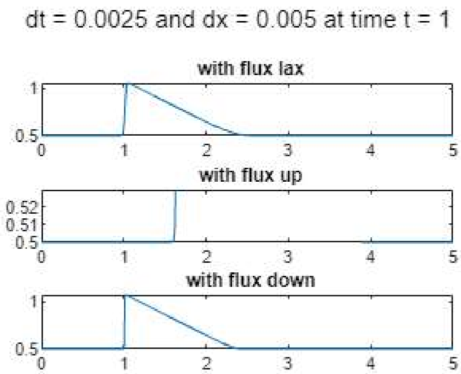



```
sumOverDomain =
sumOverDomain(:,:,1) =

  576.6867


sumOverDomain(:,:,2) =

  576.6867


sumOverDomain(:,:,3) =

  576.6867
```




```
sumOverDomain =
```

```
sumOverDomain(:,:,1) =

  576.6867


sumOverDomain(:,:,2) =

   NaN


sumOverDomain(:,:,3) =

  576.6867
```




```
sumOverDomain =
sumOverDomain(:,:,1) =

  576.6867


sumOverDomain(:,:,2) =

   NaN


sumOverDomain(:,:,3) =

  576.6867
```

We can observe that the sum over the domain $\sum_{j=0} u_j^n$ stays the same over all time steps for all the methods which doesn't diverge.

Also one can notice that that now the sharp edge at the concentration is even more visible

More efficient implementation

Matrix instead of loops