Exercise set 10: Discontinuous Galerkin

Numerical Methods for Hyperbolic Partial Differential Equations
IMATH, FS-2020
Lecturer: Dr. Philipp Öffner
Teaching Assistant: Davide Torlo

Problem 10.1 Discontinouus Galerkin (DG) (8pts)

Consider the conservation law

$$\partial_t u(x,t) + \partial_x f(u(x,t)) = 0, \quad x \in \Omega = [a,b],\, t \in \mathbb{R}^+,\, f \in \mathcal{C}^2(\mathbb{R}). \tag{1}$$

Consider a high order DG approximation. Subdivide Ω in N cells $K_j = [x_{j-1/2}, x_{j+1/2}]$, where $\{x_{j-1/2}\}_{j=1}^{N+1}$ are equispaced points and $x_{1/2} = a$ and $x_{N+1/2} = b$. Consider as finite dimensional functional space

$$V := \{v \in \mathbb{L}^2(\Omega) : v|_{K_j} \in \mathbb{P}^p(K_j)\ \forall j = 1,\dots,N\},$$

where $p \in \mathbb{N}$ is the maximum degree of the polynomial considered.

1. Write the weak formulation of the equation for an element K_j.

   Solution

   Multiply the conservation law $\partial_t u + \partial_x f(u) = 0$ by a test function $v \in V$ and integrate on a volume K_j.

   Obtain:

   $$\int_{K_j} v\partial_t u dx + \int_{K_j} v\partial_x f(u) dx = 0.$$

   Integrating by parts you obtain:

   $$\int_{K_j} v\partial_t u dx - \int_{K_j} f(u)\partial_x v dx + [f(u)v]_{x_{j-1/2}}^{x_{j+1/2}} = 0.$$

2. Let $\{\tilde{\varphi}^i(y)\}_{i=0}^p$ be a basis for $\mathbb{P}^p([-1,1])$ and, through an affine transformation, let $\{\varphi_{K_j}^i(x)\}_{i=0}^p$ be the corresponding basis for $\mathbb{P}^p(K_j)$, hence

   $$\text{span}\langle\{\varphi_{K_j}^i|_{K_j}(x), j = 1,\dots,N,\, i = 0,\dots,p\}\rangle = V.$$

   Consider the approximation solution on the cell K_j as

   $$u|_{K_j}(x,t) = u_j(x,t) = \sum_{i=0}^{p} u_{j,i}(t)\varphi_{K_j}^i(x). \tag{2}$$

   Here, with an abuse of notation we use u as the approximation solution, instead of the exact one. From now on, u will be the approximation solution. Show that the semidiscrete formulation of DG can be written as

   $$\int_{K_j} \varphi_{K_j}^i(x)\,\partial_t u(x,t) dx - \int_{K_j} f(u(x,t))\partial_x \varphi_{K_j}^i(x)\,dx + \Big[f(u(x,t))\varphi_{K_j}^i(x)\Big]_{x_{j-1/2}}^{x_{j+1/2}} = 0, \quad j = 1,\dots,N \tag{3}$$

and write and expression for $\Big[f(u(x,t))\varphi^i_{K_j}(x)\Big]^{x_{j+1/2}}_{x_{j-1/2}}$. Are the solution values at the interface well defined? How can one define these terms?

Solution

We simply use as test function the basis functions of the space V and we obtain (3). In this formulation the values at the interfaces of the cells are not well defined, indeed the two reconstructions in the two cells may not coincide there. Hence, instead of $f(u(x_{j+1/2}))$ one has to consider a numerical flux, for example Rusanov $f^{\text{num}}(u,v) = \frac{u+v}{2} - \max(|f'(u)|,|f'(v)|)\frac{v-u}{2}$, in the following way

$$\begin{aligned}&\Big[f(u(x,t))\varphi^i_{K_j}(x)\Big]^{x_{j+1/2}}_{x_{j-1/2}} = \\ &f^{\text{num}}(u_j(x_{j+1/2},t),u_{j+1}(x_{j+1/2},t))\varphi^i_{K_j}(x_{j+1/2}) - f^{\text{num}}(u_{j-1}(x_{j-1/2},t),u_j(x_{j-1/2},t))\varphi^i_{K_j}(x_{j-1/2}).\end{aligned} \tag{4}$$

3. Using explicit Euler time discretization the (3) can be rewritten as a linear system for every cell K_j

$$\sum_{i_2=0}^{p} M_{i_1,i_2}\frac{u^{n+1}_{j,i_2}-u^n_{j,i_2}}{\Delta t} = -r_{i_1}, \quad \text{for } i_1 = 0,\dots,p. \tag{5}$$

Write explicitly the definition of the mass matrix M and the right hand side r. What happens if the basis functions $\tilde{\varphi}^i$ are orthonormal?

Solution

Exploiting the weak form with explicit Euler, we obtain the following formulation

$$\sum_{i_2=0}^{p}\int_{K_j}\varphi^{i_1}_{K_j}\varphi^{i_2}_{K_j}dx\frac{u^{n+1}_{j,i_2}-u^n_{j,i_2}}{\Delta t} = -\left\{\int_{K_j}\partial_x\varphi^{i_1}_{K_j}f(u_j(x,t^n)) - \Big[f(u(x,t^n))\varphi^i_{K_j}(x)\Big]^{x_{j+1/2}}_{x_{j-1/2}}\right\}, \quad \text{for } i_1 = 0,\dots,p, \tag{6}$$

where the values at the interfaces are defined by (4). So, we can define the mass matrix as

$$M_{i_1,i_2} = \int_{K_j}\varphi^{i_1}_{K_j}\varphi^{i_2}_{K_j}dx \tag{7}$$

and the right hand side as

$$r_{i_1}(u^n) = \int_{K_j}\partial_x\varphi^{i_1}_{K_j}f(u_j(x,t^n)) - \Big[f(u(x,t^n))\varphi^i_{K_j}(x)\Big]^{x_{j+1/2}}_{x_{j-1/2}}. \tag{8}$$

4. Derive the classical FV scheme starting from the DG scheme.

Solution

In order to obtain the FV method from the DG formulation, we have to use piecewise constant basis functions $\tilde{\varphi}^0 \equiv 1$. Then the mass matrix is nothing more than

$$M_{0,0} = \int_{K_j}1dx = \Delta x. \tag{9}$$

While the right hand side simplifies, since $\partial_x\varphi^0 = 0$ and we obtain

$$r_0 = \Big[f(u(x,t^n))\varphi^i_{K_j}(x)\Big]^{x_{j+1/2}}_{x_{j-1/2}} = f^{\text{num}}(u^n_{j,0},u^n_{j+1,0}) - f^{\text{num}}(u^n_{j-1,0},u^n_{j,0}). \tag{10}$$

Summing up the terms we obtain

$$u_{j,0}^{n+1} = u_{j,0}^n - \frac{\Delta t}{\Delta x}\left(f^{\text{num}}(u_{j,0}^n, u_{j+1,0}^n) - f^{\text{num}}(u_{j-1,0}^n, u_{j,0}^n)\right). \tag{11}$$

5. Consider the linear advection equation, i.e. $f(u) = c\,u$, a central difference flux

$$\begin{aligned}\Big[f(u(x,t))\varphi_{K_j}^i(x)\Big]_{x_{j-1/2}}^{x_{j+1/2}} &= c\frac{u_j(x_{j+1/2}) + u_{j+1}(x_{j+1/2})}{2}\varphi_{K_j}^i(x_{j+1/2}) \\ &\quad - c\frac{u_{j-1}(x_{j-1/2}) + u_j(x_{j-1/2})}{2}\varphi_{K_j}^i(x_{j-1/2}),\end{aligned} \tag{12}$$

periodic boundary conditions ($x_{1/2} = x_{N+1/2}$) and the semidiscrete form (3). Prove that

$$\frac{1}{2}\frac{d}{dt}\int_\Omega u(x,t)^2 dx = 0. \tag{13}$$

Solution

Starting with

$$\partial_t \int_{K_j} u_j(x,t)\phi_{K_j}^i - a\int_{K_j} u_j \partial_x \phi_{K_j}^i + a[u(x,t)\phi_{K_j}^i]_{x_{j-1/2}}^{x_{j+1/2}} = 0, \quad 0 \le i \le p,$$

Because $u_j \in V$, we can choose $\phi_{K_j}^i = u_j$. In this case, we obtain:

$$\partial_t \int_{K_j} (u_j)^2 dx - a\frac{1}{2}\int_{K_j} \partial_x (u_j)^2 dx + a[u(x,t)u_j(x,t)]_{x_{j-1/2}}^{x_{j+1/2}} = 0,$$

where

$$\begin{aligned}&[u(x,t)u_j(x,t)]_{x_{j-1/2}}^{x_{j+1/2}} = \\ &\frac{1}{2}\left(u_j(x_{j+1/2})^2 + u_j(x_{j+1/2})u_{j+1}(x_{j+1/2})\right) - \frac{1}{2}\left(u_{j-1}(x_{j-1/2})u_j(x_{j-1/2}) + (u_j(x_{j-1/2}))^2\right).\end{aligned}$$

Note that

$$a\frac{1}{2}\int_{K_j} \partial_x (u_j)^2 dx = \frac{a}{2}\left(u_j(x_{j+1/2})^2 - u_j(x_{j-1/2})^2\right) \tag{14}$$

Then

$$\frac{1}{2}\partial_t \int_{K_j} (u_h^i)^2 dx = \{u_j(x_{j+1/2})^2 - u_j(x_{j-1/2})^2 \tag{15}$$

$$- u_j(x_{j+1/2})^2 - u_j(x_{j+1/2})u_{j+1}(x_{j+1/2}) + u_{j-1}(x_{j-1/2})u_j(x_{j-1/2}) + (u_j(x_{j-1/2}))^2\} = \tag{16}$$

$$= \frac{a}{2}\{-u_j(x_{j+1/2})u_{j+1}(x_{j+1/2}) + u_{j-1}(x_{j-1/2})u_j(x_{j-1/2})\}. \tag{17}$$

To finish, sum over all cells the quantities in (17). Since the boundary conditions are periodic and the summation is telescopic, we have:

$$\frac{1}{2}\partial_t \sum_j \int_{K_j} (u_j(x,t))^2 = 0,$$

which shows the L_2 stability.

Problem 10.2 Implementation of a DG scheme (12 pts)

In this exercise you will implement a 1-dimensional discontinuous Galerkin code for the conservation law

$$\partial_t u + \partial_x f(u) = 0$$

defined in some domain $\Omega = [a, b]$ and periodic boundary conditions. Consider a high order DG approximation given by (3) and (5).

Remember that in $\Big[f(u(x,t))\varphi^i_{K_j}(x)\Big]_{x_{j-1/2}}^{x_{j+1/2}}$ you will need to use a numerical flux to evaluate the interface values of $f(u)$, i.e.

$$f(u(x_{j+1/2},t))\varphi^i_{K_j}(x_{j+1/2}) := f^{\text{num}}(u_j(x_{j+1/2},t), u_{j+1}(x_{j+1/2},t))\varphi^i_{K_j}(x_{j+1/2}).$$

Consider nodal DG of degree $p = 2$ on Gauss Lobatto points, this means that the basis function considered are Lagrangian basis functions defined on $[-1, 1]$ as

$$\tilde{\varphi}^1 = \frac{x(x-1)}{2}, \tag{18}$$

$$\tilde{\varphi}^2 = (1+x)(1-x), \tag{19}$$

$$\tilde{\varphi}^3 = \frac{x(1+x)}{2}. \tag{20}$$

To compute the integrals (mass matrix and volume term) use a Gauss Lobatto quadrature rule on $[-1, 1]$

$$\int_{-1}^{1} g(x)dx \approx \sum_{q=1}^{3} w_q g(x_q), \quad x_q = [-1, 0, 1],\ w_q = [1/3, 4/3, 1/3]. \tag{21}$$

Hint:

- Which are the integrals that must be computed in (5)? How can they be transformed onto $[-1, 1]$? How does the quadrature rule scale for these integrals?

Add a 3rd order time integration method SSPRK3:

$$\begin{aligned}
\mathbf{u}^{(1)} &= \mathbf{u}^n + \Delta t L(\mathbf{u}^n) \\
\mathbf{u}^{(2)} &= \frac{3}{4}\mathbf{u}^n + \frac{1}{4}\mathbf{u}^{(1)} + \frac{1}{4}\Delta t L(\mathbf{u}^{(1)}) \\
\mathbf{u}^{n+1} &= \frac{1}{3}\mathbf{u}^n + \frac{2}{3}\mathbf{u}^{(2)} + \frac{2}{3}\Delta t L(\mathbf{u}^{(2)}),
\end{aligned}$$

where L is the evolution operator defined, using (5), as

$$L(\mathbf{u}) = -M^{-1}\mathbf{r}(\mathbf{u}). \tag{22}$$

Test your code for the linear advection equation $\partial_t u + \partial_x u = 0$ on two initial conditions:

1. For $x \in [-2, 2]$, $t \in [0, 1]$, with periodic boundary conditions and initial data given by:

$$u_0 = \cos(\pi x) \tag{23}$$

2. For $x \in [-2, 2]$, $t \in [0, 1]$, with periodic boundary conditions and initial data given by:

$$u_0 = \begin{cases} 1 & \text{if } -1 \le x \le 0 \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

For the first initial condition, compute the accuracy of the scheme.

What do you notice for the second initial conditions? Do you have ideas on how to fix it?

Remarks

The point of this exercise is to gain some experience in implementing an end-to-end numerical scheme. The important steps should be independently checked and debugged, in order to be sure that everything works. Feel free to follow the following suggested code structure (way more simplified than a pseudocode):

Data: initial condition, model information, solver information
Result: solution at T
$u^0_{i,j} \leftarrow$ initial condition;
while $t < T$ **do**
  $\mathbf{u}^{n+1}$=SSPRK3 ($\mathbf{u}^n$,Δt);
  $t = t + \Delta t$
end

Algorithm 1: main

Data: $u^n_{i,j}$ (all coefficients), Δt
Result: Using the method of lines, you can write the space and time update independently: $\partial_t \mathbf{u} = L(\mathbf{u})$.
Below is the time-marching algorithm given by SSPRK3:

$\mathbf{u}^{(1)} = \mathbf{u}^n + \Delta t L(\mathbf{u}^n)$;
$\mathbf{u}^{(2)} = \frac{3}{4}\mathbf{u}^n + \frac{1}{4}\mathbf{u}^{(1)} + \frac{1}{4}\Delta t L(\mathbf{u}^{(1)})$;
$\mathbf{u}^{n+1} = \frac{1}{3}\mathbf{u}^n + \frac{2}{3}\mathbf{u}^{(2)} + \frac{2}{3}\Delta t L(\mathbf{u}^{(2)})$;

Algorithm 2: SSPRK3

Data: $u_{i,j}$ (modes)
Result: Computes the evolution operator $L(\mathbf{u})$
compute volume integral $\int_{K_j} f(u_j(x))\partial_x \varphi^i_{K_j} dx$ using the quadrature rule, $j = 1, \dots, N$, $i = 0, \dots, p$;
compute numerical fluxes at the interface of the cells $f^{\text{num}}_{j-1/2}$ for $j = 1, \dots, N+1$;
compute surface integral $[f(u(x))\varphi^i_{K_j}]^{x_{j+1/2}}_{x_{j-1/2}}$ using the numerical fluxes ;
compute the final operator $-M^{-1}\mathbf{r}_{K_j}$ for every cell $j = 1, \dots, N$;

Algorithm 3: compute the evolution operator L(u)

Hints

- Note that the mass matrix doesn't change in time and for different cells, so you can compute it just once, invert it just once, store it and use it every time you need it.
- Check by hand that your mass matrix computed with that quadrature rule is correct
- Implement a visualization function that plots the solution, plotting in each interval the corresponding solution. This can be used to debug to check whether your reconstruction makes sense or not.
- For the numerical flux use Rusanov
- Note that for $p = 0$ and $\tilde{\varphi}^1(y) = 1$ you should get the 1st order FVM. You can use this fact to debug your code!

Solution

Thanks to the choice of the basis functions (Lagrangian in Gauss–Lobatto points) and the quadrature rule (Gauss–Lobatto quadrature), we have some extra properties. First of all the integrals changes a bit.

We have to compute the following integrals

$$M_{i_1,i_2} = \int_{K_j} \varphi^{i_1}_{K_j}(x)\varphi^{i_2}_{K_j}(x)dx = \frac{\Delta x}{2}\int_{-1}^{1} \tilde{\varphi}^{i_1}(y)\tilde{\varphi}^{i_2}(y)dy \tag{25}$$

where the coefficient comes from the change of variable in the integral $x = x_j + y\frac{\Delta x}{2}$, which leads to $dx = \frac{\Delta x}{2}dy$.

The second integral transform differently and we have

$$\int_{K_j} f(u_j(x,t^n))\partial_x \varphi_{K_j}^{i_1} dx = \int_{-1}^{1} f\left(\sum_{i_2} u_{j,i_2}^n \tilde{\varphi}^{i_2}(y)\right) \frac{\partial_y}{\partial_x}\partial_y \tilde{\varphi}^{i_1}(y)\frac{dx}{dy}dy = \int_{-1}^{1} f\left(\sum_{i_2} u_{j,i_2}^n \tilde{\varphi}^{i_2}(y)\right) \partial_y \tilde{\varphi}^{i_1}(y)dy, \tag{26}$$

where the affine transformation we have used before, here doesn't play a big role, because it affects in opposite ways the derivative and the integral increment.

Moreover, the mass matrix, even if the basis functions are not orthogonal, is approximated with a diagonal one thanks to the quadrature formula

$$M_{i_1,i_2} = \frac{\Delta x}{2}\int_{-1}^{1} \tilde{\varphi}^{i_1}(y)\tilde{\varphi}^{i_2}(y)dy \approx \frac{\Delta x}{2}\sum_q w_q \tilde{\varphi}^{i_1}(x_q)\tilde{\varphi}^{i_2}(x_q) = w_q \delta_{i_1,q}\delta_{i_2,q}. \tag{27}$$

In our specific case

$$M = \frac{\Delta x}{2}\begin{pmatrix} 1/3 & 0 & 0 \\ 0 & 4/3 & 0 \\ 0 & 0 & 1/3 \end{pmatrix}. \tag{28}$$

Code

To code the DG method, I have used some already existing maps to obtain Gauss–Lobatto nodes, see code 1, for any order of accuracy we want to use, and I have coded in a general way the lagrangeBasis functions, see code 2, and their derivatives, see code 3. This part was not necessary and in the code it's commented an easier implementation of the quadrature rule and of the basis functions as prescribed in the exercise.

The numerical fluxes are the ones we have used in the previous exercises, see code 4.

The time integration method is encoded in a function which returns two matrices for strong stability preserving Runge–Kutta methods, see code 5. It returns 2 matrices that can be used for the SSPRK formulation.

The core code is in code 6. We start defining the classical geometry features, the solver parameters, the geometry connectivity, the quadrature rule, the basis functions and their evaluation in the quadrature points, mass matrix and its inverse. Then we initialize the solution and we proceed with the timesteps. In the RHS function we compute the fluxes at interfaces and the volume integral, then we multiply by the inverse of the mass matrix.

In code 7 we test the algorithm for a simple case, then we run the convergence for all the different orders of accuracy (1,2,3,4), then we test the discontinuous solution and the Burgers equation.

Results

With smooth IC we have the expected accuracy see figure 1. Moreover, we can see the advantage of using a high order method in terms of computational time in figure 2. For the discontinuous IC we have some oscillations around the discontinuities, see figure 3. We can try the same method with Burgers equations and the solutions will be similar, see figure 4. In order to clip these oscillations, we should introduce some limiters.

```
1 function [x,w,P]=lglnodes(N)
2
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 %
5 % lglnodes.m
```

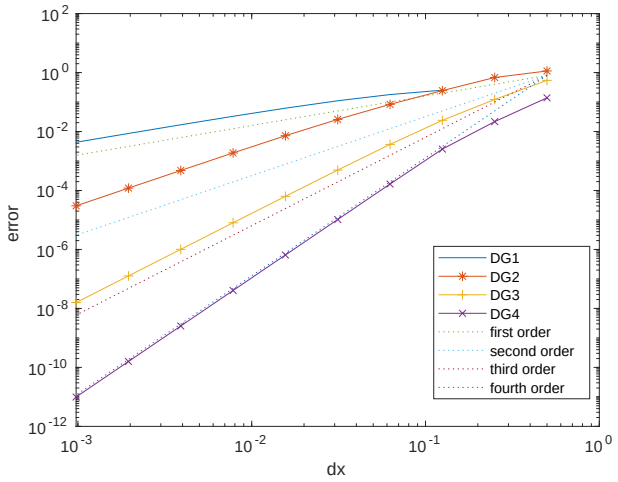

Figure 1: Convergence of DG for advection problem and smooth IC

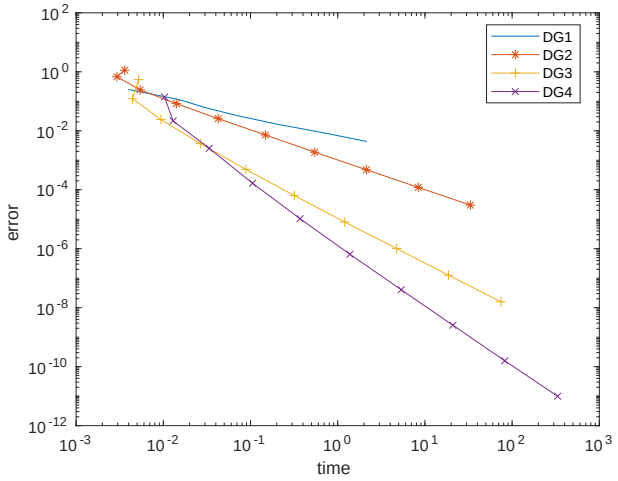

Figure 2: Convergence of DG with respect to time for advection problem and smooth IC

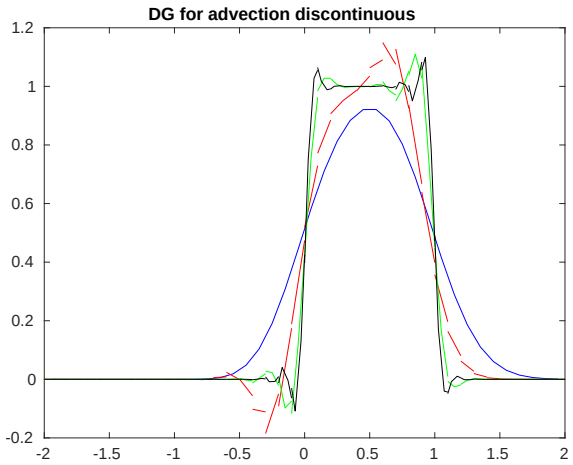

Figure 3: Simulations of DG for advection problem and discontinuous IC

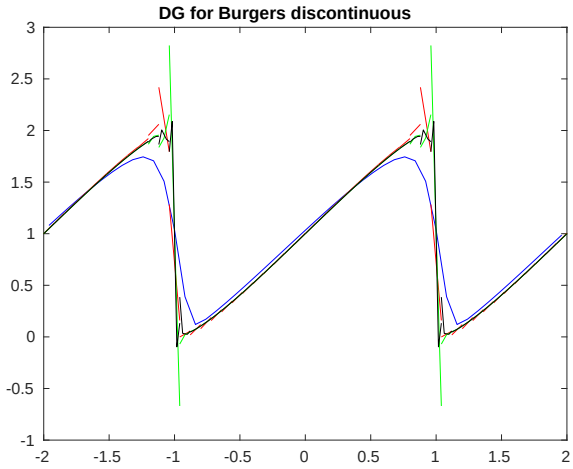

Figure 4: Simulations of DG for Burgers' equation and continuous IC

```
%
% Computes the Legendre-Gauss-Lobatto nodes, weights and the LGL Vandermonde
% matrix. The LGL nodes are the zeros of (1-x^2)*P'_N(x). Useful for numerical
% integration and spectral methods.
%
% Reference on LGL nodes and weights:
%   C. Canuto, M. Y. Hussaini, A. Quarteroni, T. A. Tang, "Spectral Methods
%   in Fluid Dynamics," Section 2.3. Springer-Verlag 1987
%
% Written by Greg von Winckel - 04/17/2004
% Contact: gregvw@chtm.unm.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if N==0
    x=0; w=2; P=1;
    return
end

% Truncation + 1
N1=N+1;

% Use the Chebyshev-Gauss-Lobatto nodes as the first guess
x=-cos(pi*(0:N)/N)';

% The Legendre Vandermonde Matrix
P=zeros(N1,N1);

% Compute P_(N) using the recursion relation
% Compute its first and second derivatives and
% update x using the Newton-Raphson method.

xold=2;

while max(abs(x-xold))>eps

    xold=x;

    P(:,1)=1;      P(:,2)=x;

    for k=2:N
        P(:,k+1)=( (2*k-1)*x.*P(:,k)-(k-1)*P(:,k-1) )/k;
    end
```

```
    x=xold-( x.*P(:,N1)-P(:,N) )./( N1*P(:,N1) );

end

w=2./(N*N1*P(:,N1).^2);
```

Listing 1: lglnodes.m

```
function [basis] = lagrangeBasis(x, pts)
%evaluation in x of lagrangian basis functions with zeros in pts
%basis is (#basis funct, dimension of x)

n=length(pts);
m=length(x);
basis=zeros([n,m]);
for k=1:n
    roots=pts([1:k-1,k+1:end]);
    pp=poly(roots);
    pp=pp/polyval(pp,pts(k));
    basis(k,:)=polyval(pp,x(:));
end




end
```

Listing 2: lagrangeBasis.m

```
function [basis] = lagrangeBasisDer(x, pts)
%evaluation in x of derivative  of lagrangian basis functions with zeros in pts
%basis is (#basis funct, dimension of x)

n=length(pts);
basis=zeros([n,size(x)]);
for k=1:n
    roots=pts([1:k-1,k+1:end]);
    pp=poly(roots);
    pp=pp/polyval(pp,pts(k));
    dd=polyder(pp);
    basis(k,:)=polyval(dd,x(:));
end




end
```

Listing 3: lagrangeBasisDer.m

```
function [fNum] = numericalFlux(scheme,f,u,v,extra)
% encode in this function different numerical fluxes
switch scheme
    case "Lax Friedrichs"
        % extra should be lambda=dt/dx
        lam=extra{1};
        fNum=(f(u)+f(v))/2-(v-u)/lam/2;
    case "Lax Wendroff"
        % extra should be lambda=dt/dx
        lam=extra{1};
        df=extra{2};
        idxn= u==v;
```

```
        idx=logical(1-idxn);
        a=zeros(size(u));
        a(idx)=(f(u(idx))-f(v(idx)))./(u(idx)-v(idx));
        fNum=(f(u)+f(v))/2-lam/2*a.*(f(v)-f(u));
    case "2stepLxW"
        % extra should be lambda=dt/dx
        lam=extra{1};
        fNum=f(0.5*(u+v) -lam/2*(f(v)-f(u)));
    case "Rusanov"
        % extra should be f'
        df=extra{1};
        fNum=(f(u)+f(v))/2-max(abs(df(u)),abs(df(v))).*(v-u)/2;
    case "Godunov"
        % extra should be omega the unique local minimum of f
        omega=extra{1}*ones(size(u));
        fNum=max(f(max(u,omega)),f(min(v,omega)));
    case "Roe"
        % extra should be empty
        idxs=u==v;
        idx=logical(1-idxs);
        fNum=f(u);
        A=(f(u(idx))-f(v(idx)))./(u(idx)-v(idx));
        fNum(idx)=f(u(idx)).*(A>=0)+f(v(idx)).*(A<0);
    case "EO" %Engquist--Osher
        % extra should be omega the unique local minimum of f
        omega=extra{1}*ones(size(u));
        fNum=f(max(u,omega))+ f(min(v,omega));
end


end
```

Listing 4: numericalFlux.m

```
function [A, B] = SSPRK(deg)
% Some Strong Stability Preserving Runge Kutta methods

if deg==1
        A=1; B=1;
elseif deg==2
        A=[1 0; 0.5, 0.5];
        B=[1 0; 0 0.5];
elseif deg==3
        A=[1 0 0; 3/4  1/4 0; 1/3 0 2/3];
        B=[1 0 0; 0 1/4 0; 0 0 2/3];
elseif deg>=4
    A=zeros(10); B=zeros(10);
    A(1:4,1:4)=eye(4); B(1:4, 1:4)=1/6*eye(4);
    A(5,1)=3/5; A(5,5)=2/5; B(5,5)=1/15;
    A(6:9,6:9)=eye(4); B(6:9, 6:9)=1/6*eye(4);
    A(10,1)=1/25; A(10,5)=9/25; A(10,10)=3/5; B(10,5)=3/50; B(10,10)=1/10;
end

end
```

Listing 5: SSPRK.m

```
function [u,xplot,t] = runNodalDG(model, solver, varargin)
if nargin<3
    withplot=0;
else
    withplot=varargin{1};
end

```

```
% geometry definitions
N=solver.Nx;                                                    %number of cells
solver.Ndof=solver.maxDegree+1;                                 % degrees of freedom in a cell
xinter=linspace(model.a,model.b,N+1);                           % x values at interfaces of cells
x_center=(xinter(1:end-1)+xinter(2:end))/2;                     % x values at center of cells
dx=xinter(2)-xinter(1); solver.dx=dx;                           % cell size
maxdfu=max(abs(model.df(model.u0(x_center))));  % max df(u) for CFL
dt=dx*solver.CFL/maxdfu;                                        % dt, for the moment CFL\leq
    1/solver.Ndof, if one doesn't want to impose it add here /solver.Ndof
Nt=ceil(model.T/dt);                                            % Number of timesteps
dt=model.T/Nt; solver.dt=dt;                                    % dt
t=linspace(0,model.T,Nt+1); solver.t=t;                         % time discretization

% geometry connectivity for periodic BC
solver.jCells=1:solver.Nx;
solver.jLCells=[N, 1:N-1];
solver.jRCells=[2:N, 1];

% Compute the quadrature points and weights, which are also the one
% defining the basis functions
% For only degree 3
% solver.qPts=[-1,0,1]; solver.qWeigth=[1/3, 4/3, 1/3];
solver.qPts=zeros(1,solver.Ndof);
solver.qWeight=zeros(1,solver.Ndof);
[solver.qPts(1,:), solver.qWeight(1,:), ~]=lglnodes(solver.maxDegree);

%%%Define basis functions
%%%Simple version for only order 3
%solver.basis=@(x) [x.*(x-1)/2; (1-x).*(1+x); (1+x).*x/2 ];
%solver.derBasis=@(x) [x-1/2; -2*x; 1/2+x];
%%%More general version
solver.basis=@(x) lagrangeBasis(x,solver.qPts);
solver.derBasis=@(x) lagrangeBasisDer(x,solver.qPts);

% Evaluate basis fct and derivatives in quadrature nodes
solver.basisQuad=solver.basis(solver.qPts);
solver.derBasisQuad=solver.derBasis(solver.qPts);

% Compute the mass matrix via quadrature rule and the inverse
solver.MM=zeros(solver.Ndof);
for m=1:solver.Ndof
    for k=1:solver.Ndof
        solver.MM(k,m)= sum(solver.basisQuad(m,:).*solver.basisQuad(k,:).*solver.qWeight);
    end
end
solver.Mi=inv(solver.MM);


% Initialization. Here we choose also how to represent the data
% un is a matrix with all the degrees of freedom for each cell of the
% domain
% Also the xplot variable follows this structure, since we are using a
% nodal DG, the plot will simply be plot(xplot,un)
un=zeros(solver.Ndof,solver.Nx);
xplot=zeros(solver.Ndof,solver.Nx);
for k=1:N
    un(:,k)=model.u0(solver.qPts*dx/2+x_center(k));
    xplot(:,k)=solver.qPts*dx/2+x_center(k);
end
solver.xplot=xplot;

u{1}=un;
if withplot
    figure(4)
```

```
71      plot(xplot,un)
        title(sprintf('DG%d time=%g',solver.Ndof,0))
73      drawnow;
        pause(0.5)
75  end


77
    extra=defineExtraScheme(solver.scheme,dt/dx,model.df);
79
    % Calling the  SSPRK matrices and initializing the stages for u and RHS
81  [RKA,RKB] = SSPRK(solver.Ndof);
    nRK=size(RKA,1)+1;
83  uRK=cell(nRK,1);
    RHSRK=cell(nRK,1);
85  for kt=2:Nt+1
        uRK{1}=un;
87      for kRK=2:nRK
            RHSRK{kRK-1}= computeRHS(uRK{kRK-1}, solver, model, extra);
89          uRK{kRK}=zeros(size(un));
            for k2=1:kRK-1
91              uRK{kRK}=uRK{kRK} + RKA(kRK-1,k2)*uRK{k2}-dt*RKB(kRK-1,k2)*RHSRK{k2};
            end
93      end
        un1 = uRK{nRK};
95
    %     RHS=computeRHS(un, solver, model, extra);
97  %     u1=un-dt*RHS;
    %     RHS1=computeRHS(u1, solver, model, extra);
99  %     u2=3/4*un+1/4*u1-1/4*dt*RHS1;
    %     RHS2=computeRHS(u2, solver, model, extra);
101 %     un1=1/3*un+2/3*u2-2/3*dt*RHS2;

103     if withplot
            figure(4)
105         plot(xplot,un1)
            title(sprintf('DG%d time=%g',solver.Ndof,t(kt)))
107         drawnow;
            pause(0.5)
109     end
        un=un1;
111     u{kt}=un1;
    end
113
    end
115

    function RHS=computeRHS(un, solver, model, extra)
117     uL=un(1,:);
        uR=un(solver.Ndof,:);
119     %%% numerical fluxes in j+1/2 ( fInter(jCell)=f_{j+1/2})
        fInter=numericalFlux(solver.scheme, model.f, uR(solver.jCells),uL(solver.jRCells), extra);
121     %%%  fluxes = phi(1)*f_{j+1/2} - phi(-1)*f_{j-1/2}
        fluxes=solver.basisQuad(:,end).*fInter(solver.jCells)-solver.basisQuad(:,1).*fInter(solver
        .jLCells);
123     %%% Here we compute the integral of der of basis fct times f(u)
        volumeInt=zeros(solver.Ndof,solver.Nx);
125     for k=1:solver.Nx
            for sigma=1:solver.Ndof
127             volumeInt(sigma, k)=sum(solver.derBasisQuad(sigma,:).*model.f(un(:,k))'.*solver.
        qWeight);
            end
129     end

131     % Finally we multiply by the inverse of the mass matrix with the
        % scaling factor of the integral increment
```

```
    RHS=2/solver.dx*solver.M1*(fluxes-volumeInt);
end

function extra=defineExtraScheme(scheme,lam,df)
switch scheme
    case {"Lax Friedrichs","2stepLxW"}
        extra={lam};
    case "Lax Wendroff"
        extra={lam, df};
    case "Rusanov"
        extra={df};
    case {"Godunov","EO"}
        extra={0}; %only Burgers
    case "Roe"
        extra={};
end

end
```

Listing 6: runNodalDG.m

```
model.f=@(u) u;%u.^2/2;
model.df=@(u) ones(size(u));
model.T=10;
model.a=-2;
model.b=2;
model.u0=@(x)  1+ 0.2* cos(pi*x);% (x<0).*(x>-1);% (x<0).*(x>-1);% (x<1).*(x>0);%uL*(x<0)+uR*(x>=0); %cos(pi*x);%cos(pi*x);
model.BC="periodic";%"dirichlet";%"periodic";

%% run DG degree 3 with plot and Rusanov flux
solver.scheme="Rusanov";%"Rusanov";%"2stepLxW";%"Lax Wendroff";%"Lax Wendroff";%"Rusanov";%"Godunov";%"Roe";%Rusanov";%"EO";%"Lax Friedrichs"
solver.Nx = 100;
solver.CFL = 0.45;
solver.maxDegree=2;
withplot=0;
[u,x,t]= runNodalDG(model, solver, withplot);
plot(x,u{end})
stop()

%% Convergence
model.f=@(u) u;
model.df=@(u) ones(size(u));
model.T=1;
model.a=-2;
model.b=2;
model.u0=@(x) 1+ 0.2* cos(pi*x);% (x<1).*(x>0); %(x<1).*(x>0);%0.2* cos(pi*x);% uL*(x<0)+uR*(x>=0); %cos(pi*x);%cos(pi*x);
model.BC="periodic";%"dirichlet";%"periodic";
model.exact=@(x,t) model.u0(x-t);

solver.maxDegree=3;
solver.CFL = 0.2;

scheme="Rusanov";%,"Lax Friedrichs"];%;"Godunov";"Roe";"EO";"Lax Wendroff";"2stepLxW"];
styles=["-","*-","+-","x-",":",".-","+"]
nn=6;
Ns=2.^[1:nn];

clear u x t ent errors times

for deg=1:4
    solver.maxDegree=deg-1;
```

```
    for n=1:nn
        solver.Nx=Ns(n);
        disp(Ns(n))
        tic
        [u,x,t]= runNodalDG(model, solver);
        times(deg,n)=toc;
        errors(deg,n)=computeError(u,x,t,model);
    end
end

fig=figure()
for deg=1:4
    loglog(1./Ns,errors(deg,:),styles(deg),'DisplayName',strcat('DG',num2str(deg)))
    hold on
end
loglog(1./Ns,1./Ns*errors(1,1)*Ns(1),':','DisplayName','first order')
loglog(1./Ns,1./Ns.^2*errors(1,1)*Ns(1)^2,':','DisplayName','second order')
loglog(1./Ns,1./Ns.^3*errors(1,1)*Ns(1)^3,':','DisplayName','third order')
loglog(1./Ns,1./Ns.^4*errors(1,1)*Ns(1)^4,':','DisplayName','fourth order')
legend('Location','best')
xlabel('dx')
ylabel('error')
saveas(fig,'errorDG.pdf')

fig=figure()
for deg=1:4
    loglog(times(deg,:), errors(deg,:),styles(deg),'DisplayName',strcat('DG',num2str(deg)))
    hold on
end
legend('Location','best')
xlabel('time')
ylabel('error')
saveas(fig,'errorvsTimeDG.pdf')

%% Discontinuous solution
model.u0=@(x)    (x<0).*(x>-1);% (x<1).*(x>0);%uL*(x<0)+uR*(x>=0); %cos(pi*x);%cos(pi*x);

withplot=0;
solver.scheme="Rusanov";%"Rusanov";%"2stepLxW";%"Lax Wendroff";%"Lax Wendroff";%"Rusanov";%"
    Godunov";%"Roe";%Rusanov";%"EO";%"Lax Friedrichs"
solver.Nx = 40;
solver.CFL = 0.2;
colors=["b","r","g","k"];

for deg=0:3
    solver.maxDegree=deg;
    [u,x,t]= runNodalDG(model, solver, withplot);
    fig=figure(5);
    plot(x,u{end},colors(deg+1))
    hold on
end
title('DG for advection discontinuous')
saveas(fig,'testDisc.pdf')

%% Burgers' equation
model.f=@(u) u.^2/2;
model.df=@(u) u;% ones(size(u));
model.T=0.5;
model.a=-2;
model.b=2;
model.u0=@(x)    1+cos(pi*x);% (x<0).*(x>-1);% (x<1).*(x>0);%uL*(x<0)+uR*(x>=0); %cos(pi*x);%
    cos(pi*x);
model.BC="periodic";%"dirichlet";%"periodic";

```

```
solver.scheme="Rusanov";%"Rusanov";%"2stepLxW";%"Lax Wendroff";%"Lax Wendroff";%"Rusanov";%"
    Godunov";%"Roe";%Rusanov";%"EO";%"Lax Friedrichs"
solver.Nx = 50;
solver.CFL = 0.2;
solver.maxDegree=3;
withplot=0;


for deg=0:3
    solver.maxDegree=deg;
    [u,x,t]= runNodalDG(model, solver, withplot);
    fig=figure(6);
    plot(x,u{end},colors(deg+1))
    hold on
end
title('DG for Burgers discontinuous')
saveas(fig,'testBurg.pdf')



function err=computeError(u,x,t,model)
    err=norm(u{end}-model.exact(x,t(end)))*sqrt(x(2)-x(1));
end
```

Listing 7: testNodalDG.m

Lax Friedrichs

You might have noticed that the Lax Friedrichs numerical flux produces an unstable scheme. I was very surprised at the beginning, but we can check why it is happening. We are considering the linear advection equation $\partial_t u + \partial_x u = 0$ and we have noticed instabilities. What we can do is to use the von Neumann analysis, with some extra attention. We use the ansatz that, as usual, moving by one cell we can obtain the solution just by a multiplication times $e^{ik\Delta x}$ where k is the wave number we consider as initial state:

$$\begin{cases} u_{j+1,0} = e^{ik\Delta x} u_{j,0} \\ u_{j+1,1} = e^{ik\Delta x} u_{j,1} \\ u_{j+1,2} = e^{ik\Delta x} u_{j,2}. \end{cases} \tag{29}$$

In this case the amplification factor becomes an amplification matrix

$$\begin{pmatrix} u_{j,0}^{n+1} \\ u_{j,1}^{n+1} \\ u_{j,2}^{n+1} \end{pmatrix} = A \begin{pmatrix} u_{j,0}^{n} \\ u_{j,1}^{n} \\ u_{j,2}^{n} \end{pmatrix} \tag{30}$$

If we consider a simple explicit Euler time integration, we can see that the matrix A can be computed in the following way using the definition of the exercise.

$$\begin{pmatrix} u_{j,0}^{n+1} \\ u_{j,1}^{n+1} \\ u_{j,2}^{n+1} \end{pmatrix} = \begin{pmatrix} u_{j,0}^{n} \\ u_{j,1}^{n} \\ u_{j,2}^{n} \end{pmatrix} - \frac{\Delta t}{\Delta x} \underbrace{\begin{pmatrix} 6 & 0 & 0 \\ 0 & 3/2 & 0 \\ 0 & 0 & 6 \end{pmatrix}}_{M^{-1}} \begin{pmatrix} -f_{j-1/2} - \left(-\frac{3}{6}u_{j,0}^n - \frac{4}{6}u_{j,1}^n + \frac{1}{6}u_{j,2}^n\right) \\ -\left(\frac{2}{3}u_{j,0}^n - \frac{2}{3}u_{j,2}^n\right) \\ f_{j+1/2} - \left(-\frac{1}{6}u_{j,0}^n + \frac{4}{6}u_{j,1}^n + \frac{3}{6}u_{j,2}^n\right) \end{pmatrix}.$$

Where we have used the definitions of the basis functions and the quadrature rules. Now, we have to define the numerical flux and we will compare Lax Friedrichs (reminder $f' = 1$)

$$f_{j+1/2} = \frac{u_{j,2} + u_{j+1,0}}{2} - \frac{\Delta x}{\Delta t} \frac{u_{j+1,0} - u_{j,3}}{2} \tag{31}$$

and Rusanov

$$f_{j+1/2} = \frac{u_{j,2} + u_{j+1,0}}{2} - \frac{u_{j+1,0} - u_{j,2}}{2} = u_{j,2}. \tag{32}$$

Using these definitions we obtain for Rusanov (which coincide with upwind)

$$\begin{pmatrix} u_{j,0}^{n+1} \\ u_{j,1}^{n+1} \\ u_{j,2}^{n+1} \end{pmatrix} = \begin{pmatrix} u_{j,0}^{n} \\ u_{j,1}^{n} \\ u_{j,2}^{n} \end{pmatrix} + \frac{\Delta t}{\Delta x} M^{-1} \begin{pmatrix} -1/2u_{j,0}^n - 2/3u_{j,1}^n - 1/6u_{j,2}^n - e^{i\theta} \\ 2/3u_{j,0}^n - 2/3u_{j,2}^n \\ -1/6u_{j,0}^n + 2/3u_{j,1}^n - 1/2u_{j,2}^n \end{pmatrix} = (\mathtt{Id} + \lambda M^{-1} A) \begin{pmatrix} u_{j,0}^{n} \\ u_{j,1}^{n} \\ u_{j,2}^{n} \end{pmatrix}, \tag{33}$$

where

$$A = \begin{pmatrix} -\frac{1}{2} & -\frac{2}{3} & \frac{1}{6} + e^{-i\theta} \\ \frac{2}{3} & 0 & -\frac{2}{3} \\ -\frac{1}{6} & \frac{2}{3} & -\frac{1}{2} \end{pmatrix}. \tag{34}$$

For Lax Friedrichs

$$A = \begin{pmatrix} -\frac{1}{2\lambda} & -\frac{2}{3} & \frac{1}{6} + \frac{1+1/\lambda}{2} e^{-i\theta} \\ \frac{2}{3} & 0 & -\frac{2}{3} \\ -\frac{1}{6} - \frac{1-1/\lambda}{2} e^{i\theta} & \frac{2}{3} & -\frac{1}{2\lambda} \end{pmatrix} \tag{35}$$

and Central differences $f^{\text{num}}(u,v) = \frac{f(u)+f(v)}{2}$ we have

$$A = \begin{pmatrix} 0 & -\frac{2}{3} & \frac{1}{6} + \frac{1}{2} e^{-i\theta} \\ \frac{2}{3} & 0 & -\frac{2}{3} \\ -\frac{1}{6} - \frac{1}{2} e^{i\theta} & \frac{2}{3} & 0 \end{pmatrix}. \tag{36}$$

Defining as always $\lambda = \Delta t/\Delta x$. We can study the only space discretization and using the ansatz $u(x,t) = e^{\alpha t} e^{ikx}$, we have that the space discretization is stable if α is nonpositive. Hence $\alpha u = M^{-1}/\Delta x A u$ verify this condition if the eigenvalues of A are ≤ 0. For the space time discretization we have to compute the evolution matrix, for explicit Euler it is $B = (\mathtt{Id} + \lambda M^{-1} A)$, for more sophisticated time integration schemes one can take more complicated matrices and check whether the eigenvalues are inside the unit circle.

In code 8 we code the matrices and we substitute some values for λ and many values of $\theta \in [0, 2\pi]$. Check what you obtain with the different configurations.

```
%% upwind only space discretization du/dt=Au. stable if eig(A)<=0
A=@(t) [-1/2 -2/3 1/6+exp(-1i*t);...
    2/3 0 -2/3;...
    -1/6 2/3 -1/2];
D=@(t) eig(A(t));
tt=linspace(0,2*pi,100);

fig=figure()
yy=zeros(size(tt));
for k=1:3
    for z=1:length(tt)
        tmp=D(tt(z));
        yy(z)=tmp(k);
    end
    plot(real(yy),imag(yy),'x-','DisplayName', sprintf('Eig%d',k))
    hold on
end
plot([0,0],[-2,2],':','DisplayName','Stability')
legend('Location','best')
title(sprintf('upwind only space'))
saveas(fig,'stabilityUPwindspace.pdf')

```

```
% Explicit Euler u^{n+1}= B u^n  stable if |eig(B)|<=1
B=@(t,l) eye(3) + l*diag([6 3/2 6])*A(t);

D=@(t,l) eig(B(t,l));
for ll=[0.01, 0.1, 0.2]
tt=linspace(0,2*pi,100);

fig=figure()
for k=1:3
    for z=1:length(tt)
        tmp=D(tt(z),ll);
        yy(z)=tmp(k);
    end
    plot(real(yy),imag(yy),'DisplayName', sprintf('Eig%d',k))
    hold on
end
plot(sin(tt),cos(tt),':','DisplayName','Stability')
legend('Location','best')
title(sprintf('Upwind explicit euler with CFL=%g',ll))
saveas(fig,'stabilityExplicitEulerupwind.pdf')

end


%SSPRK3
Id=eye(3);
L=@(t,l) l*diag([6 3/2 6])*A(t);
U1=@(t,l) Id +L(t,l);
U2=@(t,l) 3/4*Id+1/4*U1(t,l)+1/4*L(t,l)*U1(t,l);
UU=@(t,l) 1/3*Id+2/3*U2(t,l)+2/3*L(t,l)*U2(t,l);


D=@(t,l) eig(UU(t,l));
for ll=[0.01, 0.1, 0.2, 0.4, 0.45, 0.5]
tt=linspace(0,2*pi,1000);

fig=figure()
for k=1:3
    for z=1:length(tt)
        tmp=D(tt(z),ll);
        yy(z)=tmp(k);
    end
    plot(real(yy),imag(yy),'+','DisplayName', sprintf('Eig%d',k))
    hold on
end
plot(sin(tt),cos(tt),':','DisplayName','Stability')
legend('Location','best')
title(sprintf('Upwind SSPRK3 with CFL=%g',ll))
saveas(fig,'stabilitySSPRK3upwind.pdf')

end


%% Central differences only space discretization du/dt=Au. stable if eig(A)<=0
close all

A=@(t) [0 -2/3 1/6+1/2*exp(-1i*t);...
    2/3 0 -2/3;...
    -1/6-1/2*exp(1i*t) 2/3 0];
D=@(t) eig(A(t));
tt=linspace(0,2*pi,100);

fig=figure()
yy=zeros(size(tt));
```

```
for k=1:3
    for z=1:length(tt)
        tmp=D(tt(z));
        yy(z)=tmp(k);
    end
    plot(real(yy),imag(yy),'x-','DisplayName', sprintf('Eig%d',k))
    hold on
end
plot([0,0],[-2,2],':','DisplayName','Stability')
legend('Location','best')
title(sprintf('CD only space'))
saveas(fig,'stabilityCDspace.pdf')


% Explicit Euler u^{n+1}= B u^n  stable if |eig(B)|<=1
B=@(t,l) eye(3) + l*diag([6 3/2 6])*A(t);

D=@(t,l) eig(B(t,l));
for ll=[0.01, 0.1, 0.2]
tt=linspace(0,2*pi,100);

fig=figure()
for k=1:3
    for z=1:length(tt)
        tmp=D(tt(z),ll);
        yy(z)=tmp(k);
    end
    plot(real(yy),imag(yy),'DisplayName', sprintf('Eig%d',k))
    hold on
end
plot(sin(tt),cos(tt),':','DisplayName','Stability')
legend('Location','best')
title(sprintf('CD explicit euler with CFL=%g',ll))
saveas(fig,'stabilityExplicitEulerCD.pdf')

end


%SSPRK3
Id=eye(3);
L=@(t,l) l*diag([6 3/2 6])*A(t);
U1=@(t,l) Id +L(t,l);
U2=@(t,l) 3/4*Id+1/4*U1(t,l)+1/4*L(t,l)*U1(t,l);
UU=@(t,l) 1/3*Id+2/3*U2(t,l)+2/3*L(t,l)*U2(t,l);


D=@(t,l) eig(UU(t,l));
for ll=[0.01, 0.1, 0.2]
tt=linspace(0,2*pi,1000);

fig=figure()
for k=1:3
    for z=1:length(tt)
        tmp=D(tt(z),ll);
        yy(z)=tmp(k);
    end
    plot(real(yy),imag(yy),'DisplayName', sprintf('Eig%d',k))
    hold on
end
plot(sin(tt),cos(tt),':','DisplayName','Stability')
legend('Location','best')
title(sprintf('CD SSPRK3 with CFL=%g',ll))
saveas(fig,'stabilitySSPRK3CD.pdf')
```

```
end


%% Lax Friedrichs only space discretization du/dt=Au. stable if eig(A)<=0
A=@(t,l) [-1/2/l -2/3 1/6+1/2*(1+1/l)*exp(-1i*t);...
    2/3 0 -2/3;...
    -1/6-1/2*(1-1/l)*exp(1i*t) 2/3 -1/2/l];

D=@(t,l) eig(A(t,l));
tt=linspace(0,2*pi,100);

yy=zeros(size(tt));
for ll=[0.01,0.1,0.2]
fig=figure()
    for k=1:3
        for z=1:length(tt)
            tmp=D(tt(z),ll);
            yy(z)=tmp(k);
        end

        plot(real(yy),imag(yy),'x-','DisplayName', sprintf('Eig%d',k))
        hold on
    end
    plot([0,0],[-2,2],':','DisplayName','Stability')
    legend('Location','best')
    title(sprintf('LxF only space CFL=%g',ll))
    saveas(fig,'stabilityLxFspace.pdf')
end

% Explicit Euler u^{n+1}= B u^n  stable if |eig(B)|<=1
B=@(t,l) eye(3) + l*diag([6 3/2 6])*A(t,l);

D=@(t,l) eig(B(t,l));
for ll=[0.01, 0.1, 0.2]
tt=linspace(0,2*pi,100);

fig=figure()
for k=1:3
    for z=1:length(tt)
        tmp=D(tt(z),ll);
        yy(z)=tmp(k);
    end
    plot(real(yy),imag(yy),'DisplayName', sprintf('Eig%d',k))
    hold on
end
plot(sin(tt),cos(tt),':','DisplayName','Stability')
legend('Location','best')
title(sprintf('LxF explicit euler with CFL=%g',ll))
saveas(fig,'stabilityExplicitEulerLxF.pdf')

end


%SSPRK3
Id=eye(3);
L=@(t,l) l*diag([6 3/2 6])*A(t,l);
U1=@(t,l) Id +L(t,l);
U2=@(t,l) 3/4*Id+1/4*U1(t,l)+1/4*L(t,l)*U1(t,l);
UU=@(t,l) 1/3*Id+2/3*U2(t,l)+2/3*L(t,l)*U2(t,l);


D=@(t,l) eig(UU(t,l));
for ll=[0.01, 0.1, 0.2]
tt=linspace(0,2*pi,100);
```

```
216
fig=figure()
218 for k=1:3
    for z=1:length(tt)
220     tmp=D(tt(z),ll);
        yy(z)=tmp(k);
222 end
    plot(real(yy),imag(yy),'+-','DisplayName', sprintf('Eig%d',k))
224 hold on
end
226 plot(sin(tt),cos(tt),':','DisplayName','Stability')
legend('Location','best')
228 title(sprintf('LxF SSPRK3 with CFL=%g',ll))
saveas(fig,'stabilitySSPRK3LxF.pdf')
230
end
```

Listing 8: stabAnalysis.m

Organiser: Davide Torlo, Office: home (davide.torlo@math.uzh.ch)

Published: May 14, 2020

Due date: May 28, 2020, h10.00 (use the upload tool of my.math.uzh.ch, see wiki.math.uzh.ch/public/student_upload_homework or if you have troubles send me an email).