Lab06: Multidimensional finite difference schemes for hyperbolic equations

1 Approaches to discretization in multiple spatial dimensions

1.1 Semi-discretization

Semi-discretization of the consdervation equation

$$\boldsymbol{q}_t + \boldsymbol{f}(\boldsymbol{q})_x + \boldsymbol{g}(\boldsymbol{q})_y = 0$$

leads to the linearized ODE system

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{Q} = -\mathbf{A}\,\boldsymbol{Q} - \mathbf{B}\,\boldsymbol{Q}, \mathbf{A} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}, \mathbf{B} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{q}}.$$

that can be integrated in time using standard schemes. The methods are similar to those obtained in the one spatial dimension case. As an example, we'll consider the midpoint (leap-frog scheme)

$$\boldsymbol{Q}^{n+1} = \boldsymbol{Q}^{n-1} - \mathbf{A}\,\boldsymbol{Q}^n - \mathbf{B}\,\boldsymbol{Q}^n.$$

1.2 Taylor series

Taylor series expansion to second-order accuracy:

$$\begin{aligned}
\boldsymbol{q}(t+k) &= \boldsymbol{q} + k\,\boldsymbol{q}_t + \frac{k^2}{2}\boldsymbol{q}_{\mathrm{tt}} \\
\boldsymbol{q}_t &= -\mathbf{A}\,\boldsymbol{q}_x - \mathbf{B}\,\boldsymbol{q}_y \\
\boldsymbol{q}_{tt} &= -\mathbf{A}(-\mathbf{A}\,\boldsymbol{q}_x - \mathbf{B}\,\boldsymbol{q}_y)_x - \mathbf{B}(-\mathbf{A}\,\boldsymbol{q}_x - \mathbf{B}\,\boldsymbol{q}_y)_y \\
\boldsymbol{q}_{tt} &= \mathbf{A}^2\,\boldsymbol{q}_{xx} + (\mathbf{A}\mathbf{B} + \mathbf{B}\mathbf{A}\,)\boldsymbol{q}_{xy} + \mathbf{B}^2\,\boldsymbol{q}_{yy}
\end{aligned}$$

1.3 Dimensional splitting

The linearized equation leads to the solution

$$\boldsymbol{q}(t+k,x,y) = e^{-(\mathcal{A}+\mathcal{B})k}\boldsymbol{q}(t,x,y), \mathcal{A} = \mathbf{A}\,\partial_x, \mathcal{B} = \mathbf{B}\,\partial_y.$$

The formal solution can be approximated by

$$\boldsymbol{q}(t+k,x,y) \cong e^{-\mathcal{A}k}e^{-\mathcal{B}k}\boldsymbol{q}(t,x,y),$$

or to higher order (Strang-splitting)

$$\boldsymbol{q}(t+k,x,y) \cong e^{-\mathcal{B}k/2}e^{-\mathcal{A}k}e^{-\mathcal{B}k/2}\boldsymbol{q}(t,x,y)$$

2 Implementation

2.1 Global definitions

A module is constructed with global definitions.

```
MODULE Global
  INTEGER, PUBLIC, PARAMETER :: sgl = SELECTED_REAL_KIND( 7,16)
  INTEGER, PUBLIC, PARAMETER :: dbl = SELECTED_REAL_KIND(14,32)
  INTEGER, PUBLIC, PARAMETER :: qPrec = dbl, xPrec = dbl
  INTEGER, PUBLIC, PARAMETER :: FTCS=1, Upwind=2, LaxFriedrichs=3, &
          LeapFrog=4, LaxWendroff=5, BeamWarming=6
END MODULE Global
```

`SELECTED_REAL_KIND(P,R)` is a Fortran intrinsic function that returns the available type closest to the requested precision of `P` decimal digits and an exponent range `R`. Two, possibly different, precisions are defined for the dependent variables (q) and the independent variables (space, time).

2.2 Time stepping

```
SUBROUTINE scheme(method,m,nSteps,cfl,Q0,Q1,Q,Qlft,Qrgt)
  USE Global
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: method,m,nSteps
  REAL(KIND=xPrec), INTENT(IN) :: cfl
  REAL(KIND=qPrec), DIMENSION(0:m+1), INTENT(INOUT) :: Q0,Q1
  REAL(KIND=qPrec), DIMENSION(0:m+1), INTENT(OUT) :: Q
  REAL(KIND=qPrec), DIMENSION(0:nSteps), INTENT(IN) :: Qlft,Qrgt
```

A common interface to all finite difference schemes:
method: scheme to apply
m: number of interior nodes
nSteps: number of time steps
cfl: CFL number
Q0,Q1: initial conditions
Q: final state after time stepping
Qlft: boundary values at left
Qrgt: boundary values at right

```
  INTEGER mm1,mp1,n
  REAL(KIND=xPrec) :: hcfl, hcfl2
```

Internal variable declarations

```
  mm1=m-1; mp1=m+1; hcfl=cfl/2; hcfl2=cfl**2/2
```

Precomputation of common expresions

```
  SELECT CASE (method)
```

Start of SELECT statement

2.2.1 FTCS

```
    CASE (FTCS)
      DO n=1,nSteps
        IF (cfl>0) THEN
          Q0(0) = Qlft(n-1); Q0(mp1) = Q0(m)
        ELSE
          Q0(0) = Q0(1); Q0(mp1) = Qrgt(n-1)
        END IF
        Q(1:m) = Q0(1:m) - hcfl*(Q0(2:mp1) - Q0(0:mm1))
        Q0(1:m) = Q(1:m)
      END DO
```

Carry out time steps.
For $u>0$, use specified left boundary value, extrapolate at right. For $u<0$ use specified right boundary value, extrapolate at left.

$Q_i^{n+1} = Q_i^n - \frac{\nu}{2}(Q_{i+1}^n - Q_{i-1}^n)$

Stability. Use Von Neumann analysis $Q_j^n \sim G^n e^{ij\theta}$, with $\theta = \xi h$.

```
In[45]:= Q[n_,j_,theta_] = G[n] Exp[I j theta]
```

$G(n)\, e^{ij\theta}$

```
In[46]:= FTCS = Q[n+1,j,theta] - (Q[n,j,theta] - nu/2 (Q[n,j+1,theta]-Q[n,j-1,theta]))
```

$\frac{1}{2}\nu\,(G(n)\,e^{i(j+1)\theta} - G(n)\,e^{i(j-1)\theta}) + G(n)\,(-e^{ij\theta}) + G\,(n+1)\,e^{ij\theta}$

```
In[47]:= expr = Expand[FTCS/G[n]] /. G[n+1]/G[n]->A
```

$A\,e^{ij\theta} - \frac{1}{2}\nu\,e^{i(j-1)\theta} + \frac{1}{2}\nu\,e^{i(j+1)\theta} - e^{ij\theta}$

```
In[48]:= AmpFact[theta_] = Simplify[ComplexExpand[A /. Solve[expr == 0,A][[1,1]]]]
```

$1 - i\,\nu\sin(\theta)$

```
In[49]:=
```

The amplifcation factor is $|A| \geqslant 1$, hence the FTCS method is always unstable.

Modified equation.

```
In[10]:= FTCS = s[t+k,x] - (s[t,x] - nu/2 (s[t,x+h]-s[t,x-h]))
```

$\frac{1}{2}\nu\,(s\,(t,h+x) - s\,(t,x-h)) + s\,(k+t,x) - s(t,x)$

```
In[14]:= mFTCS = Simplify[Normal[Series[FTCS,{k,0,2},{h,0,2}]]]
```

$h\,\nu\,s^{(0,1)}(t,x) + \frac{1}{2}k^2\,s^{(2,0)}(t,x) + k\,s^{(1,0)}(t,x)$

From above series expansions find the modified equation

$$s_t + u s_x = -\frac{k}{2}s_{tt} + \mathcal{O}(k^2,h^2) = -\frac{u^2 k}{2}s_{xx} + \mathcal{O}(k^2,h^2),$$

an advection-diffusion equation with negative diffusivity, again indicating instability of the FTCS method.

<pre>CASE (Upwind)
  DO n=1,nSteps
    IF (cfl>0) THEN
      Q0(0) = Qlft(n-1); Q0(mp1) = Q0(m)
      Q(1:m) = Q0(1:m) - cfl*(Q0(1:m) - Q0(0:mm1))
    ELSE
      Q0(0) = Q0(1); Q0(mp1) = Qrgt(n-1)
      Q(1:m) = Q0(1:m) - cfl*(Q0(2:mp1) - Q0(1:m))
    END IF
    Q0(1:m) = Q(1:m)
  END DO</pre>
	$Q_i^{n+1} = Q_i^n - \nu(Q_i^n - Q_{i-1}^n)$ for $u > 0$
	
	$Q_i^{n+1} = Q_i^n - \nu(Q_{i+1}^n - Q_i^n)$ for $u < 0$

Stability. Use Von Neumann analysis $Q_j^n \sim G^n e^{ij\theta}$, with $\theta = \xi h$ (assume $u > 0$)

```
In[87]:= Upwind = Q[n+1,j,theta] - (Q[n,j,theta] - nu (Q[n,j,theta]-Q[n,j-1,theta]))
```

$\nu\left(G(n)\,e^{ij\theta} - G(n)\,e^{i(j-1)\theta}\right) + G(n)\left(-e^{ij\theta}\right) + G\left(n+1\right)e^{ij\theta}$

```
In[88]:= expr = Expand[Upwind/G[n]] /. G[n+1]/G[n]->A
```

$A\,e^{ij\theta} - \nu\,e^{i(j-1)\theta} + \nu\,e^{ij\theta} - e^{ij\theta}$

```
In[89]:= AmpFact[theta_] = Simplify[TrigReduce[ComplexExpand[A /. Solve[expr == 0,A][[1,1]]]]]
```

$-i\,\nu\sin(\theta) + \nu\cos(\theta) - \nu + 1$

```
In[90]:= A[theta_,nu_]=Sqrt[TrigReduce[ComplexExpand[AmpFact[theta] Conjugate[AmpFact[theta]]]]]
```

$\sqrt{-2\,\nu^2\cos(\theta) + 2\,\nu^2 + 2\,\nu\cos(\theta) - 2\,\nu + 1}$

```
In[93]:= UpwindStabPlot=Plot[Table[A[theta,nu],{nu,0.1,1.0,0.1}],{theta,0,2Pi},
           GridLines->Automatic,Axes->False,Frame->True,FrameLabel->{"theta","A(theta)"}];
         Export["/home/student/courses/MATH761/lab05/UpwindStabPlot.pdf",UpwindStabPlot]
```

/home/student/courses/MATH761/lab05/UpwindStabPlot.pdf

```
In[94]:=
```

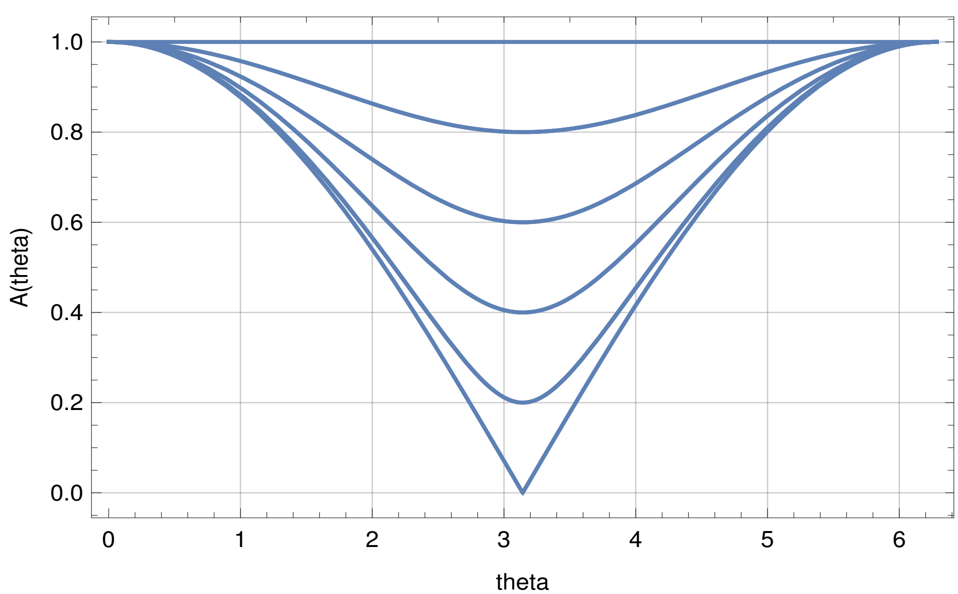

Figure 1. Upwind amplification factor obtained from Von Neumann analysis

Modified equation.

```
In[94]:= Upwind = s[t+k,x] - (s[t,x] - nu (s[t,x]-s[t,x-h]))
```

$\nu\left(s(t,x) - s\left(t,x-h\right)\right) + s\left(k+t,x\right) - s(t,x)$

```
In[96]:= mUpwind = Simplify[Normal[Series[Upwind,{k,0,2},{h,0,2}]]]
```

$-\frac{1}{2}h^2\,\nu\,s^{(0,2)}(t,x) + h\,\nu\,s^{(0,1)}(t,x) + \frac{1}{2}k^2\,s^{(2,0)}(t,x) + k\,s^{(1,0)}(t,x)$

In[97]:=

From above series expansions find the modified equation

$$s_t + u s_x = -\frac{k}{2} s_{tt} + \frac{h^2 \nu}{2k} s_{xx} + \mathcal{O}(k^2, h^2) = \frac{1}{2}\left(-k u^2 + \frac{h^2 \nu}{k}\right) s_{xx} + \mathcal{O}(k^2, h^2) = \frac{\nu h^2}{2k}(1-\nu) s_{xx} + \mathcal{O}(k^2, h^2),$$

again an advection-diffusion equation with diffusivity

$$\alpha = \frac{\nu h^2}{2k}(1-\nu).$$

Note that the diffusivity is zero at CFL $\nu = 1$. The analytical solution to

$$\begin{cases} s_t + u s_x = \alpha s_{xx} \\ s(t=0, x) = H(-x) \end{cases}$$

is

$$s(t,x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-ut}{\sqrt{4\alpha t}}\right)\right], \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-\xi^2} \mathrm{d}\xi$$

In[99]:= `s[t_,x_]=(1+Erf[(x-u t)/Sqrt[4 alpha t]])/2`

$$\frac{1}{2}\left(\operatorname{erf}\left(\frac{x - t u}{2\sqrt{\alpha t}}\right) + 1\right)$$

In[100]:= `Simplify[ D[s[t,x],t] + u D[s[t,x],x] - alpha D[s[t,x],{x,2}] ]`

0

In[101]:=

```
    CASE (LeapFrog)
      DO n=1,nSteps
        IF (cfl>0) THEN
          Q0(0) = Qlft(n-1); Q1(0) = Qlft(n)
          Q0(mp1) = Q1(m); Q1(mp1) = Q1(m)
        ELSE
          Q0(0) = Q0(1); Q1(1) = Q1(0)
          Q0(mp1) = Qrgt(n-1); Q0(mp1) = Qrgt(n)
        END IF
        Q(1:m) = Q0(1:m) - cfl*(Q1(2:mp1) - Q1(0:mm1))
        Q0(1:m) = Q1(1:m); Q1(1:m) = Q(1:m)
      END DO
```

$Q_i^{n+1} = Q_i^{n-1} - \nu(Q_{i+1}^n - Q_{i-1}^n)$

```
    CASE (LaxWendroff)
      DO n=1,nSteps
        IF (cfl>0) THEN
          Q0(0) = Qlft(n-1); Q0(mp1) = Q1(m)
        ELSE
          Q0(0) = Q0(1); Q0(mp1) = Qrgt(n-1); Q0(mp1) = Qrgt(n)
        END IF
        Q(1:m) = Q0(1:m) - hcfl*(Q0(2:mp1) - Q0(0:mm1)) + &
                 hcfl2*(Q0(2:mp1)-2*Q0(1:m)+Q0(0:mm1))
        Q0(1:m) = Q(1:m)
      END DO
```

$Q_i^{n+1} = Q_i^{n-1} - \frac{\nu}{2}(Q_{i+1}^n - Q_{i-1}^n) + \frac{\nu^2}{2}(Q_{i+1}^n - 2Q_i^n + Q_{i-1}^n)$

```
  END SELECT
END SUBROUTINE scheme
```

3 Numerical experiments

Compilation of the above implementation leads to a Python-loadable module than can be used for numerical experiments.

```
Python] from pylab import *
Python] import os,sys
        os.chdir('/home/student/courses/MATH761/lab05')
        cwd=os.getcwd()
        sys.path.append(cwd)
Python] from lab05 import *
Python] print scheme.__doc__
   q = scheme(method,cfl,q0,q1,qlft,qrgt,[m,nsteps])

   Wrapper for ``scheme``.

   Parameters
   ----------
   method : input int
   cfl : input float
   q0 : in/output rank-1 array('d') with bounds (m + 2)
   q1 : in/output rank-1 array('d') with bounds (m + 2)
   qlft : input rank-1 array('d') with bounds (nsteps + 1)
   qrgt : input rank-1 array('d') with bounds (nsteps + 1)

   Other Parameters
   ----------------
   m : input int, optional
       Default: (len(q0)-2)
   nsteps : input int, optional
       Default: (len(qlft)-1)

   Returns
   -------
   q : rank-1 array('d') with bounds (m + 2)
Python]
```

3.1 Discontinuous boundary condition

Study now the behavior for a discontinuous (shock) initial condition $q(0,x)=H(-x)$. The upwind solution will be compared to the exact solution $q(t,x)=H(ut-x)$ for the advection equation $q_t+uq_x=0$, and the exact solution

$$s(t,x)=\frac{1}{2}\left[1+\operatorname{erf}\left(\frac{ut-x}{\sqrt{4\alpha t}}\right)\right], \operatorname{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-\xi^2}\,\mathrm{d}\xi$$

to the modified equation $s_t+us_x=\alpha s_{xx}$, with artificial diffusivity

$$\alpha=\frac{\nu h^2}{2k}(1-\nu).$$

```
Python] def f(x,kappa):
          return zeros(size(x))
        def g(t,kappa):
          return (sign(t)+1)/2
        u=1;
Python] FTCS=1; Upwind=2; LaxFriedrichs=3; LeapFrog=4; LaxWendroff=5; BeamWarming=6;
Python] kappa=4
Python]
```

3.1.1 Upwind

```
Python] method=Upwind
        from math import erf
        m=99; h=1./(m+1); x=arange(m+2)*h;
        dcfl=0.2; tfinal=0.5;
        clf();
        qex=g(tfinal-x/u,kappa);
        plot(x[1:m],qex[1:m],'k.'); s = zeros(size(x));
        for cfl in arange(dcfl,1+dcfl,dcfl):
          k=h*cfl/u; nSteps=ceil(tfinal/k); t=arange(nSteps+1)*k; kappa=4;
          Q0=f(x,kappa); Q1=zeros(size(x));
          Qlft=g(t,kappa); Qrgt=zeros(size(t));
          Q1=scheme(method,cfl,Q0,Q1,Qlft,Qrgt);
          alpha = max(cfl*h**2*(1-cfl)/(2*k),10**(-6));
          y = (u*tfinal-x)/sqrt(4*alpha*tfinal);
          for i in range(m+1):
            s[i] = 0.5*(1+erf(y[i]));
          plot(x[1:m],Q1[1:m],'b',x[1:m],s[1:m],'r.');
        xlabel('x'); ylabel('q'); title('Shock propagation by upwind scheme');
        savefig("Lab05UpwindShockInitCond.pdf");
Python]
```

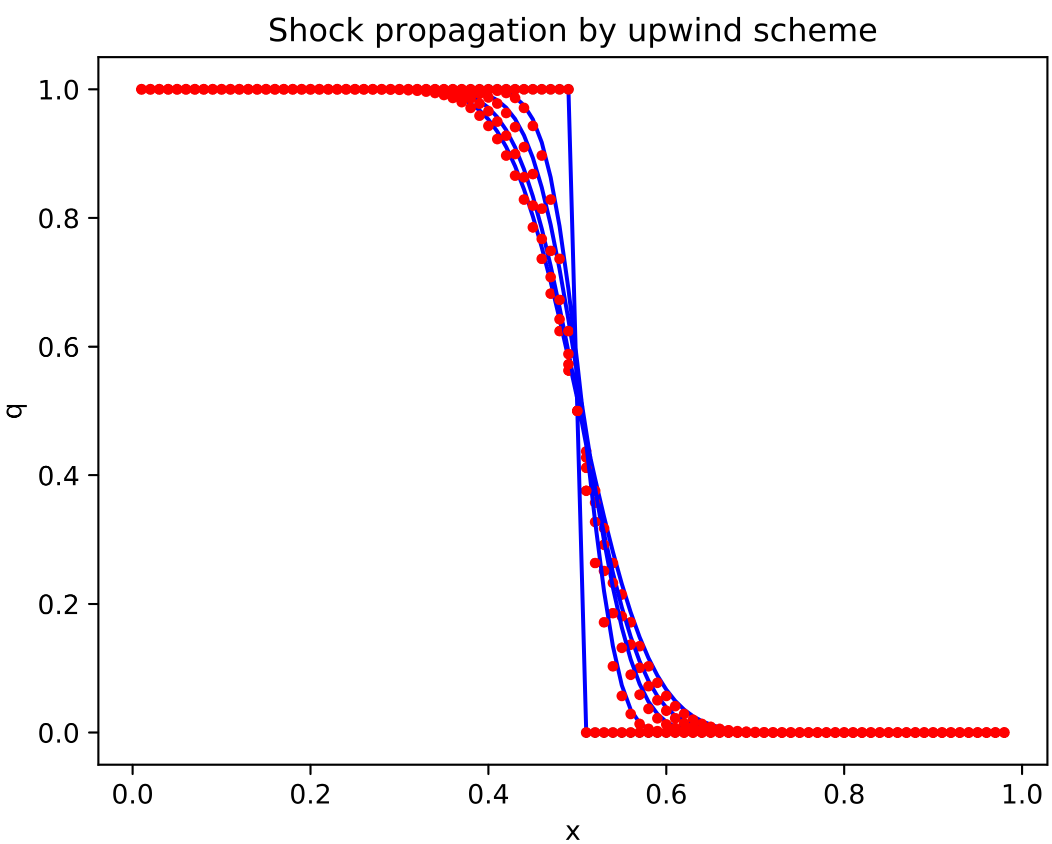

Figure 2. The exact solution to the modified equation corresponds closely to the results from the upwind scheme