Adwekcja i równanie Burgersa

Karol Janik, Michał Kazanecki

17 grudnia 2020

1 Wstęp

Podstawowym równaniem mechaniki płynów jest równanie ciągłości:

$$\frac{\partial \rho(\mathbf{x},t)}{\partial t} + \vec{\bigtriangledown} \cdot (\rho \mathbf{v}(\mathbf{x},t)) = 0 \tag{1}$$

Gdzie: ρ - gęstość cieczy, v - prędkość płynu.

Równanie to mówi, że zmiany gęstości cieczy w pewnej objętości wynikają z różnicy ilości cieczy, która wpłynęła do objętości i tej, która z niej wypłynęła. Dla przepływu jednowymiarowego ze stałą prędkością $v = c$, równanie to przyjmuje postać:

$$\frac{\partial \rho}{\partial t} + c\frac{\partial \rho}{\partial x} = 0 \tag{2}$$

Jest ono znane jako równanie adwekcji. Równanie to opisuje unoszenie pewnej substancji przez przepływający płyn. Nietrudno sprawdzić, że dowolna funkcja w postaci fali biegnącej: $u(x,t) = f(x-ct)$, spełnia to równanie.

Innym fundamentalnym równaniem mechaniki płynów jest równanie Burgersa:

$$\frac{\partial u}{\partial t} + \epsilon u\frac{\partial u}{\partial x} = 0 \tag{3}$$

$$\frac{\partial u}{\partial t} + \epsilon\frac{\partial (u^2/2)}{\partial x} = 0 \tag{4}$$

Mozna na nie spojrzeć jak na równanie adwekcji, w którym prędkość fali $c = \epsilon u$ jest proporcjonalna do amplitudy. Powoduje to zmianę kształtu fali w czasie. Wyższe części fali będą przemiaeszczać się na front, w efekcie tworząc ostry brdzeg, nazywany falą uderzeniową.

W niniejszej pracy przedsawione zostaną numeryczne rozwiązania wyżej wymienionych równań. Głowna uwaga skupia się na równaniu Burgersa, dla którego porównane zostaną dwie metody rozwiązania.

2 Algorytmy numeryczne

Sekcja ta zawiera przedstawienie wykorzystanych w pracy metod numerycznych - metody Laxa-wendroffa i algorytmu skokowego. W dodatkach znajduje się ich implementacja w języku Python.

2.1 Metoda Laxa-Wendroffa dla równania adwekcji

Użycie algorytmu skokowego dla równania adwekcji może prowadzić do niestabilnych rozwiązań. Lepsze efekty daje w tym przypadku metoda Laxa-Wendroffa, która wykorzystuje różnice drugiego rzędu dla pochodnej czasowej:

$$u(x, t + \Delta t) \approx u(x, t) + \frac{\partial u}{\partial t}\Delta t + \frac{1}{2}\frac{\partial^2 u}{\partial t^2}\Delta t^2 \tag{5}$$

Dokładne wyprowadzenie wzoru dla tej metody znaleźć można w [2]. Przyjmuje on następującą postać:

$$u_{i,j+1} = \frac{1}{2}\beta(1+\beta)u_{i-1,j} + (1-\beta^2)u_{i,j} - \frac{1}{2}\beta(1-\beta)u_{i+1,j} \tag{6}$$

Gdzie $\beta = c\frac{\Delta t}{\Delta x}$ jest numerem Couranta, indeks i oznacza kolejne punkty w przestrzeni, a indeks j kolejne punkty w czasie.

2.2 Algorytm skokowy dla równania Burgersa

Wyrażając pochodne przy pomocy różnic centralnych otrzymujemy algorytm skokowy:

$$u(x, t + \Delta t) \approx u(x, t - \Delta t) - \beta\left[\frac{u^2(x + \Delta x, t) - u^2(x - \Delta x, t)}{2}\right] \tag{7}$$

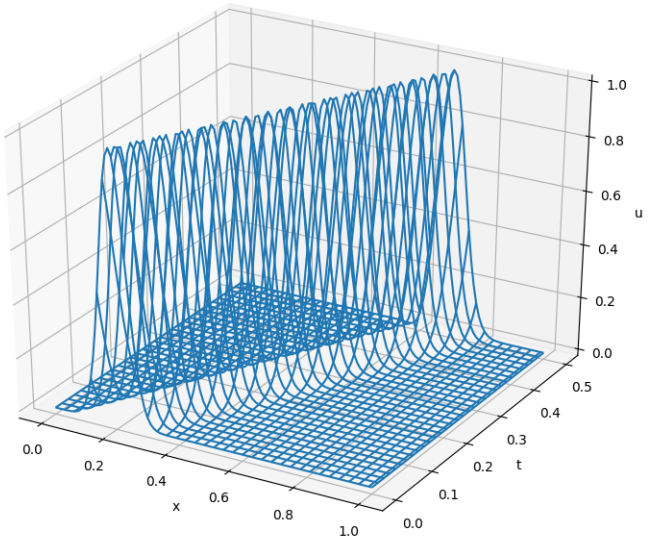
$$u_{i,j+1} = u_{i,j-1} - \beta\left[\frac{u^2_{i+1,j} - u^2_{i-1,j}}{2}\right], \beta = \frac{\epsilon}{\Delta x/\Delta t} \tag{8}$$

Gdzie β jest numerem Couranta. Dla stabilości rozwiązania wymagane jest aby $\beta < 1$. Metoda ta może prowadzić do niestabilnych rozwiązań równania Burgersa ze względu na duże różnice między wychyleniami kolejnych punktów. Lepszą stabilność osiąga się wykorzystując metodę Laxa-Wendroffa.

2.3 Metoda Laxa-Wendroffa dla równania Burgersa

Podobnie jak wcześniej, wykorzystuje się tu róznice drugiego rzędu. Dokładne wyprowadzenie wzoru znaleźć można w [1]. Ostatecznie przybiera on postać:

$$\begin{aligned} u_{i,j+1} = u_{i,j} - \frac{\beta}{4}\left(u^2_{i+1,j} - u^2_{i-1,j}\right) + \\ + \frac{\beta^2}{8}\left[(u_{i+1,j} + u_{i,j})\left(u^2_{i+1,j} - u^2_{i,j}\right) - (u_{i,j} + u_{i-1,j})\left(u^2_{i,j} - u^2_{i-1,j}\right)\right] \end{aligned} \tag{9}$$



Rysunek 1: Rozwiązanie numeryczne równania adwekcji

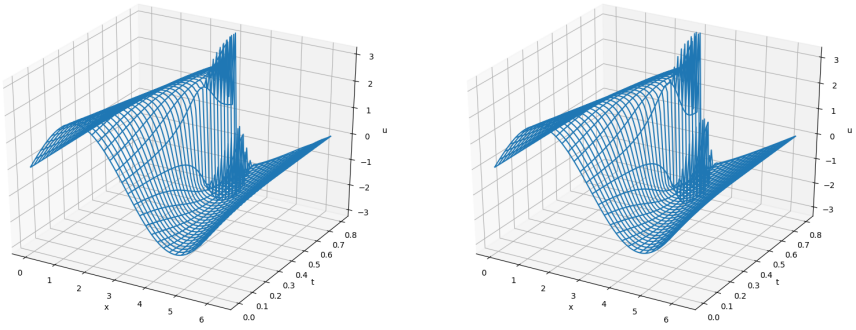
3 Wyniki

Na rysunku 1 zobaczyć można rozwiązanie równania adwekcji dla początkowego kształtu w postaci rozkładu Gaussa. Wychylenie zostało tu przedsawione w funkcji czasu i położenia. Tak jak się spodziewaliśmy, jest to fala biegnąca w kierunku dodatnim osi x. implementację algorytnu znaleźć można w dodatku A.

Rysuni 2 - 4 przedstawiają rozwiązanie równania Burgersa metodami skokowymi i Laxa-Wendroffa dla rosnącej wartości liczby Couranta. Początkowy ksztalt sinusoidalny z biegiem czasu zmienia się, tworząc ostry brzeg - falę uderzeniową. Dla $\beta = 0.2$ oba rozwiązania są bardzo podobne. W okolicy ostrego brzegu pojawiają się oscylacje, są to artefakty numeryczne. Przy rosnącej liczbie Couranta oscylacje te stają się coraz mniej regularne, szczególnie w przypadku algorytmu skokowego. Ponad to dla algorytmu skokowego pojawiają się one wcześniej i występują na całym przedziale x.

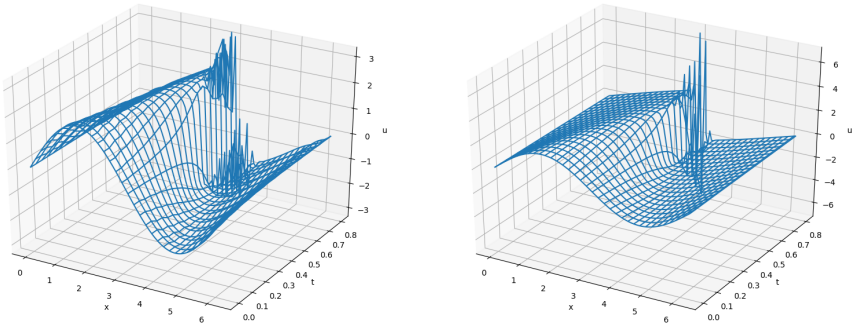
4 Podsumowanie

W pracy udało się skutecznie zaimplementować rozwiązania numeryczne równania adwekcji i równania Burgersa. W przypdaku równania Burgersa mogliśmy zaobserwować spadek stabilności wraz ze wzrostem liczby Couranta. Było to szczególnie widoczne dla algorytmu skokowego.



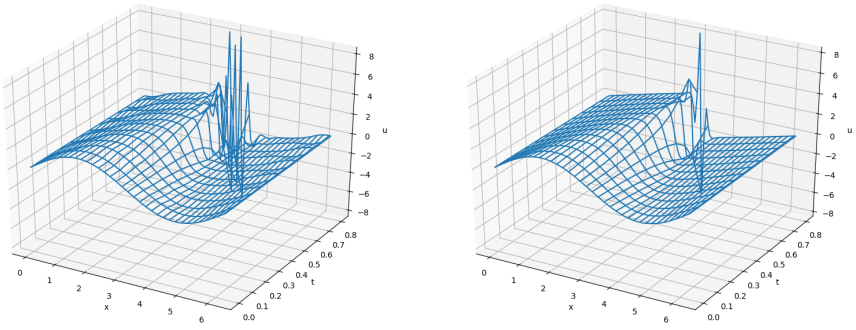
(a) Algorytm skokowy (b) Lax-Wendroff

Rysunek 2: Rozwiązanie numeryczne równania Burgersa, $\beta = 0.2$



(a) Algorytm skokowy (b) Lax-Wendroff

Rysunek 3: Rozwiązanie numeryczne równania Burgersa, $\beta = 0.6$



(a) Algorytm skokowy (b) Lax-Wendroff

Rysunek 4: Rozwiązanie numeryczne równania Burgersa, $\beta = 1$

A Równanie adwekcji

```
import numpy as np

m = 100; c = 1.
dx = 1./m; x = np.linspace(0,1,m+1)
beta = 0.8
# beta = c*dt / dx
dt = beta*dx / c
Tfinal = 0.5
n = int(Tfinal/dt); t = np.linspace(0,Tfinal,n+1)
u = np.zeros((n+1, m+1), float); u[0] = np.exp(-300.*(x-0.2)**2)

for j in range(0, n):
    for i in range(0, m-1):
        u[j+1][i+1] = (1. - beta*beta)*u[j][i+1]
        -(0.5*beta)*(1.-beta)*u[j][i+2] +(0.5*beta)*(1. + beta)*u[j][i]
```

B Równanie Burgersa

```
import numpy as np

m = 100; dx = 2*np.pi/m; x = np.linspace(0,2*np.pi,m+1)
beta = 0.2; eps = 1; dt = beta*dx/eps; T_final = 0.8
n = int(T_final/dt); t = np.linspace(0,T_final,n+1)

#Algorytm skokowy
u0 = 2*np.sin(x); u=np.zeros((n+1, m+1), float)
u[0]=u0; u[1]=u0
for j in range(1,n):
    for i in range(1,m):
        u[j+1][i]=u[j-1][i]-beta*0.5*((u[j][i+1])**2-(u[j][i-1])**2)

#Metoda Laxa-Wendroffa
w0 = 2*np.sin(x); w=np.zeros((n+1, m+1), float)
w[0]=w0
for j in range(0,n):
    for i in range(1,m):
        w[j+1][i]=w[j][i]-beta*0.25*((w[j][i+1])**2-(w[j][i-1])**2)
        +(beta**2)*0.125*((w[j][i+1]+w[j][i])*((w[j][i+1])**2-(w[j][i])**2)
                         -(w[j][i]+w[j][i-1])*((w[j][i])**2-(w[j][i-1])**2))
```

Literatura

[1] Landau R. H., Paez M. J., Bordeinau C. C., *A Survey of Computational Physics*, Princeton University Press 2012

[2] https://encyclopediaofmath.org/wiki/Lax-Wendroff_method (Dostęp 16.12.2020)