Modelowanie Procesów Fizycznych – Laboratorium 3 i 4

Model Taylora

Sylwester Macura

1) Wstęp

Celem laboratoriów była symulacja transportu znacznika w rzece. Aby to zasymulawać musimy rozwiazać adwekcyjno dyspersyjne równanie transportu (model Taylora). Do tego wykorzystamy dwie metody, metode QUICKEST (metoda jawna) oraz metode Cranka-Nicolsona (metoda niejawna).

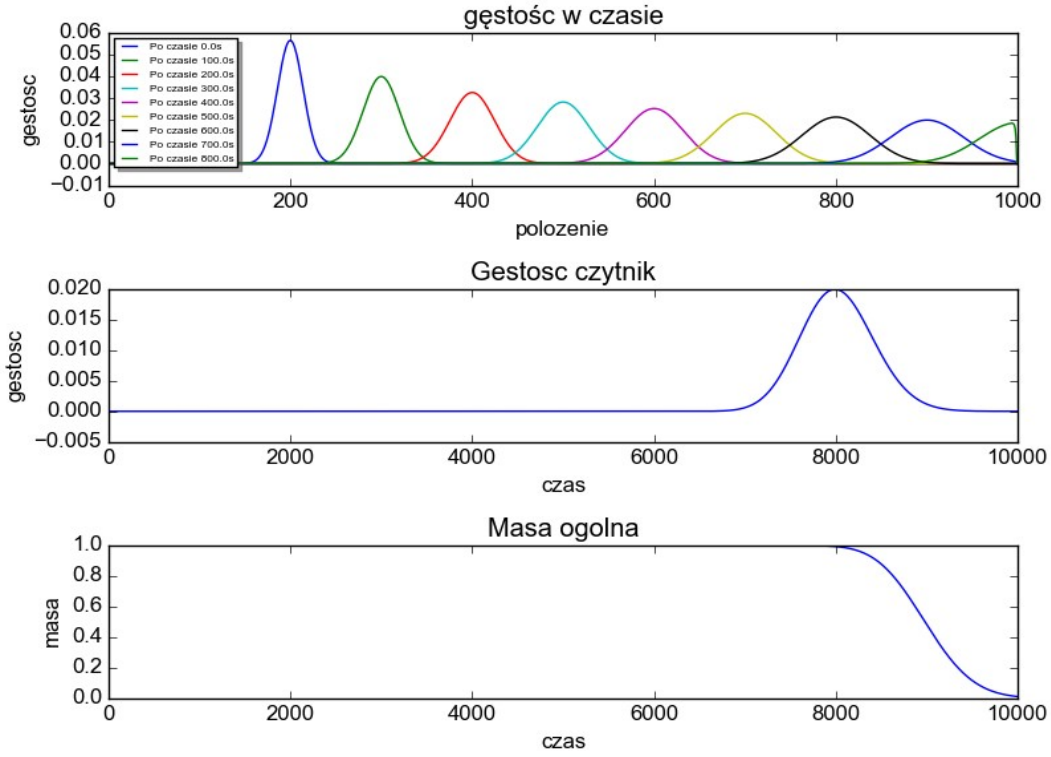
2) Model Taylora – metoda jawna

Do rozwiązania równania transportu użyjemy metody QUICKEST. Jest to metoda jawna czyli taka która do wyliczenia wyników nastepnej iteracji wykorzystuje dane poprzedniej.

2.1 Implementacja

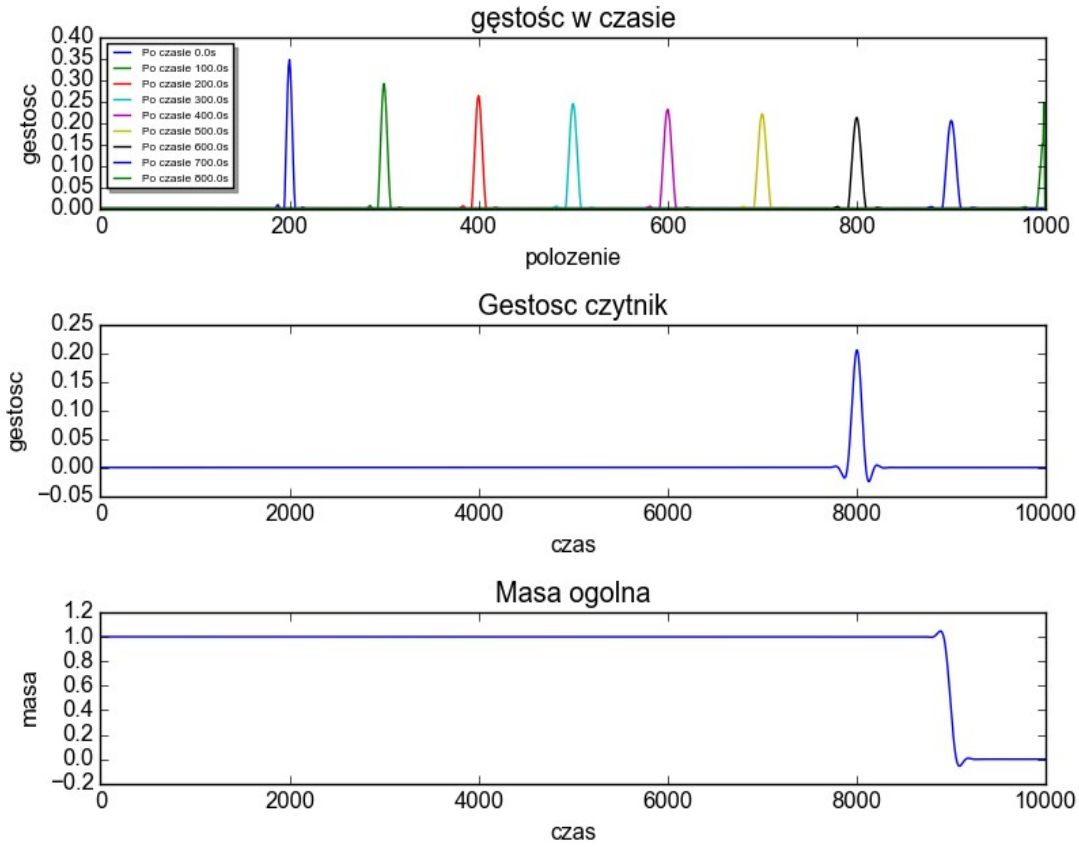
```
import pylab as P
import numpy as np
import matplotlib
#implementacja metody
def iteruj(dane, Ca, Cd):
    dane[2:-1] = dane[2:-1] + (Cd * (1.0 - Ca) - Ca / 6.0 * (Ca ** 2.0 - 3.0 *
Ca + 2.0)) * dane[3:] - (Cd * (
        2.0 - 3.0 * Ca) - Ca / 2.0 * (Ca ** 2.0 - 2.0 * Ca - 1.0)) * dane[2:-1]
+ (Cd * (1.0 - 3.0 * Ca) - Ca / 2.0 * (
        Ca ** 2.0 - Ca - 2.0)) * dane[1:-2] + (Cd * Ca + Ca / 6.0 * (Ca ** 2.0 -
1.0)) * dane[:-3]
#ustawienie parametrów podstawowych
dlugos = 100
szerokosc = 5
glebokosc = 1
matplotlib.rc('font', family='Arial')
dx = 0.1
dt = 0.1
c = P.zeros(int(dlugos / dx))
c[:] = 0.0
i = int(10 / dx)
d = int(90 / dx)
c[i] = 1.0 / (dx * szerokosc * glebokosc)
n = int(1000 / dt)
Ca = 0.1 * dt / dx
Cd = 0.01 * dt / (dx ** 2.0)
z = []
ro = []
f, axarr = P.subplots(3, 1)
subResults = []
#główna pętla
for x in range(10000):
    #iteracja
    iteruj(c, Ca, Cd)
    #pomiar wartości przy detektorze
    z.append(c[d])
    #sprawdzenie masy
    ro.append(P.sum(c[:] * (dx * szerokosc * glebokosc)))
    #pobranie wyników pośrednich
    if x % 1000 == 0:
        subResults.append(np.copy(c))
#prezentacja wyników pośrednich
for i,x in enumerate(subResults[1:]):
    axarr[0].plot(x,label="Po czasie {0}s".format(i*1000*dt))
axarr[0].set_title(u'gęstośc w czasie')
axarr[0].set_xlabel('polozenie')
axarr[0].set_ylabel('gestosc')
axarr[0].legend(loc='upper left', shadow=True,prop={'size':6})
#prezentacja wyników z detektora
axarr[1].plot(z)
axarr[1].set_title('Gestosc czytnik')
axarr[1].set_xlabel('czas')
axarr[1].set_ylabel('gestosc')
#prezentacja zachowania masy
axarr[2].plot(ro)
axarr[2].set_title('Masa ogolna')
axarr[2].set_xlabel('czas')
axarr[2].set_ylabel('masa')
f.tight_layout(pad=0.4, h_pad=1.0)
P.savefig("zad1.png")
```

2.2 Wynik Metody



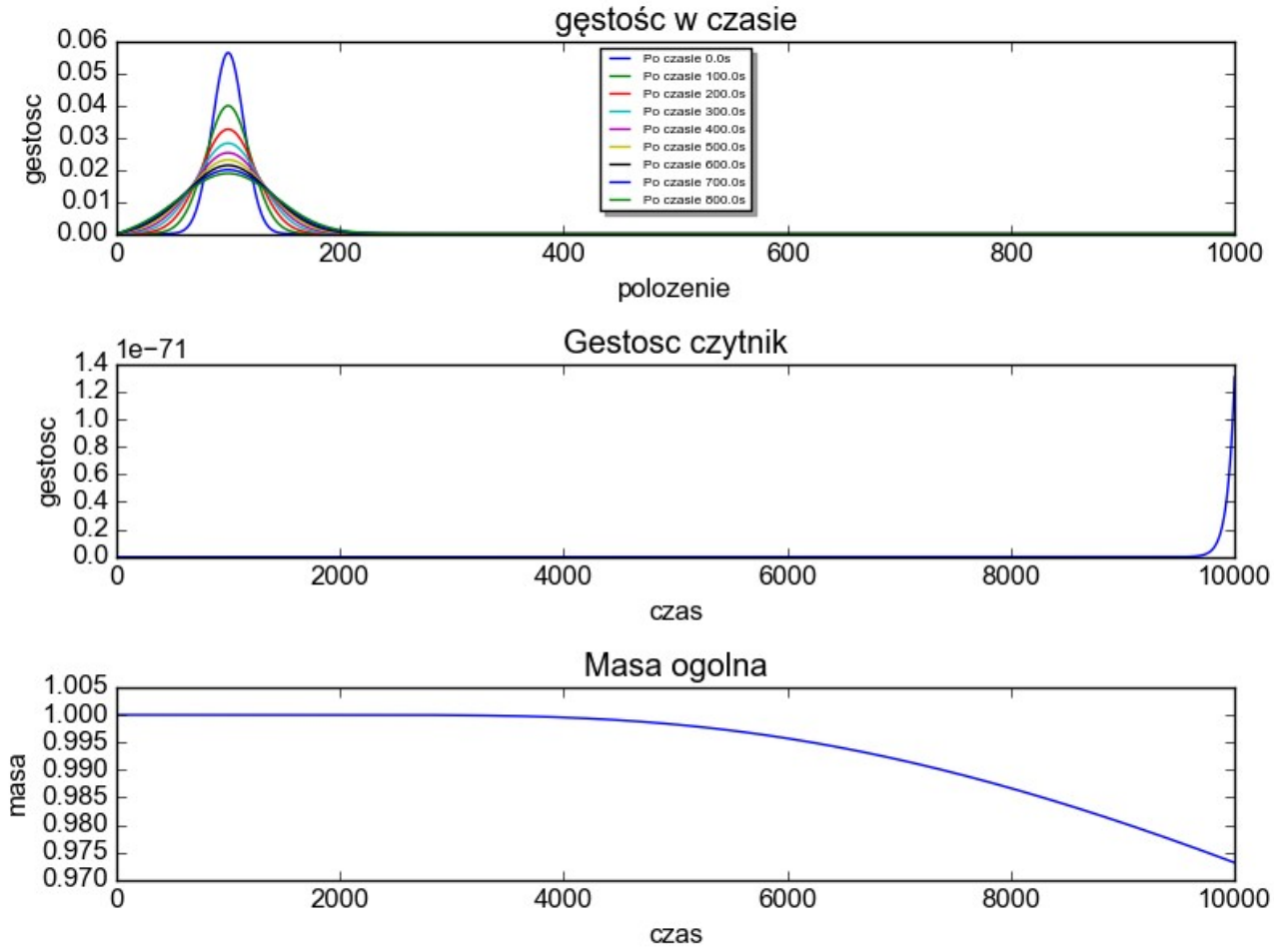
Ilustracja 1: Prezentacja wyników metody

2.3 Wriant z samą adwekcją



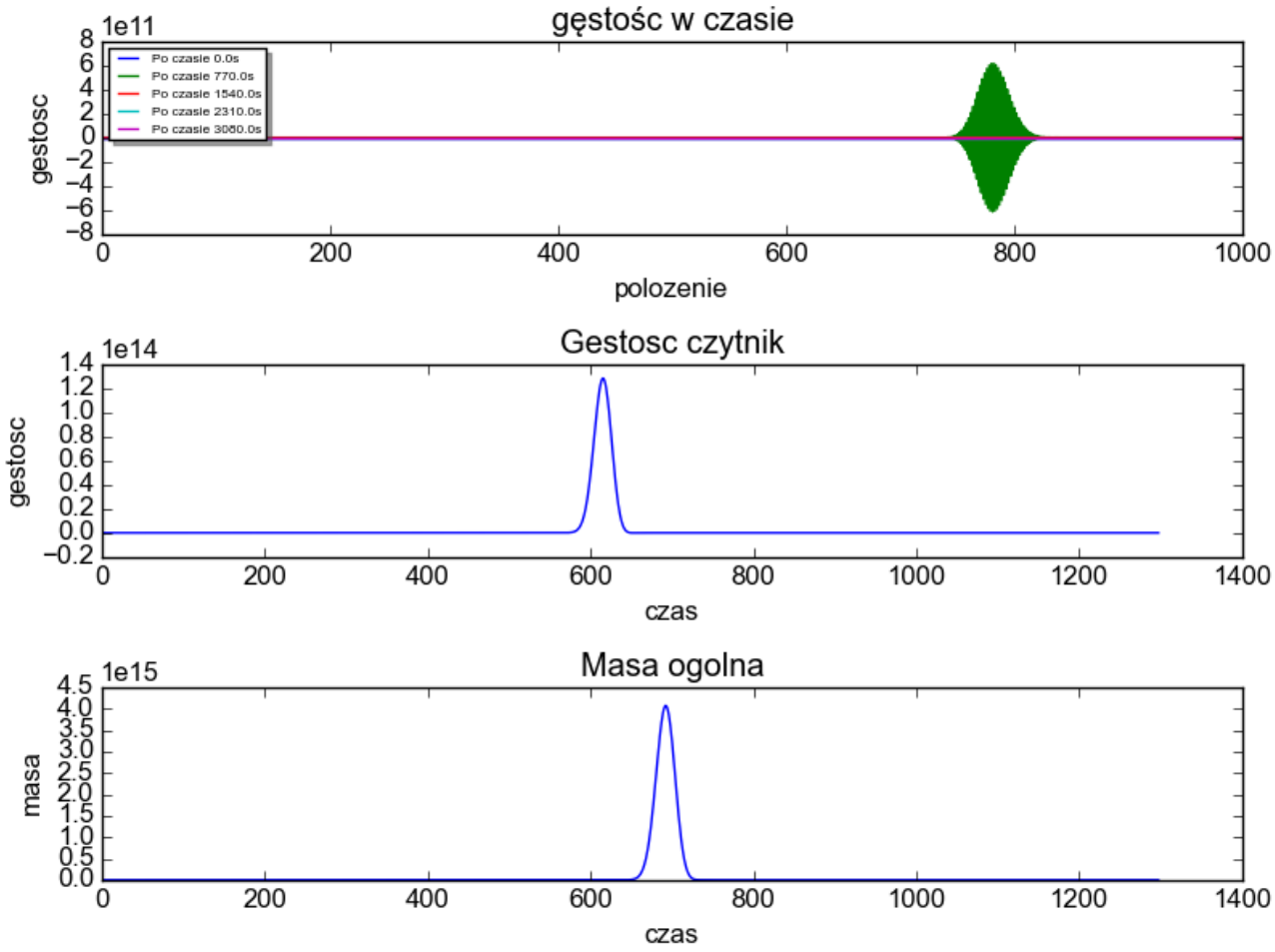
Ilustracja 2: Metoda taylora tylko adwekcja

2.4 Wariant z samą dyspersją



Ilustracja 3: Metoda Taylora tylko dyspersja

2.5 Graniczny krok czasowy



Ilustracja 4: Wynik dla dt=0.77

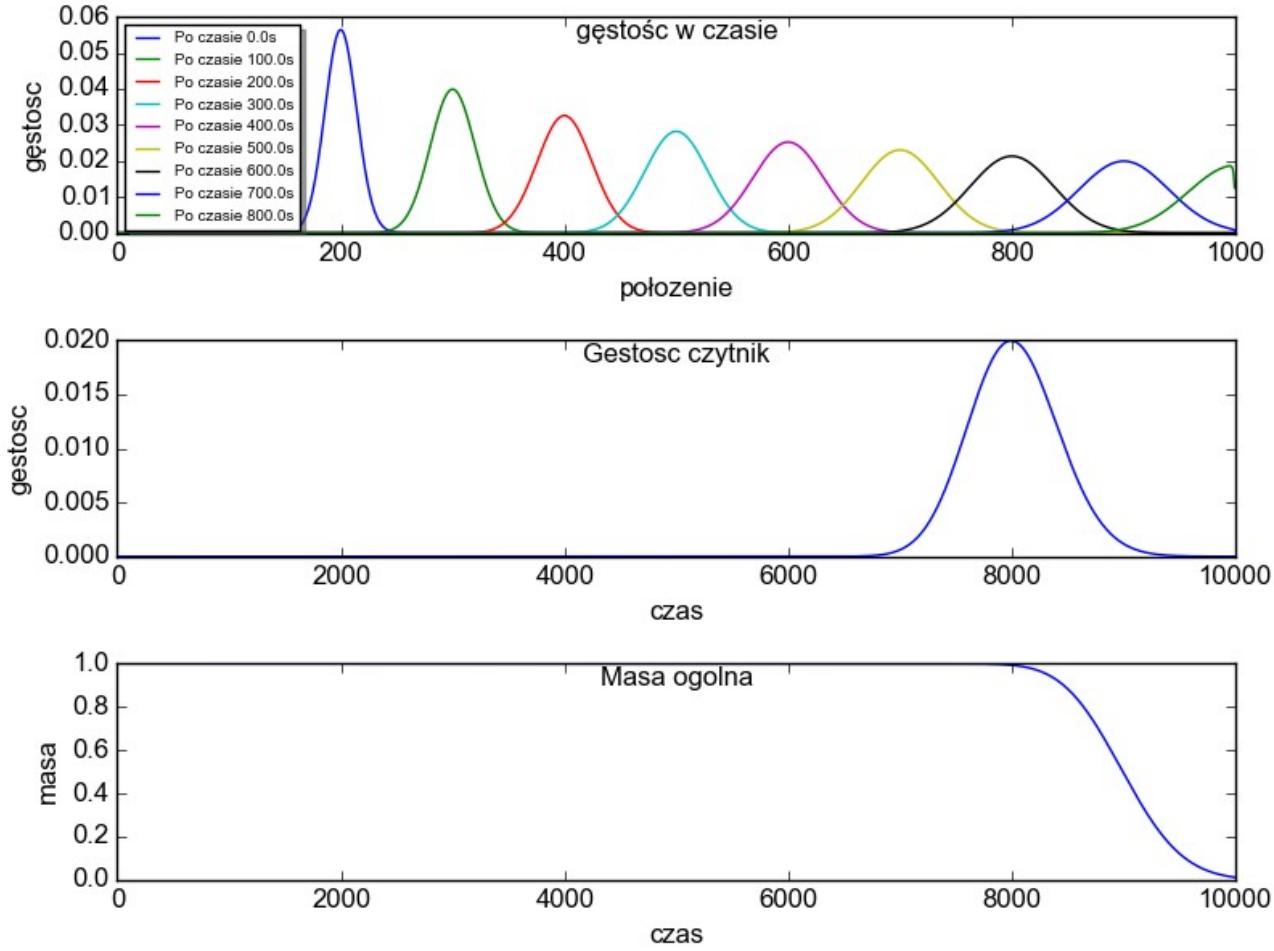
3) Model Taylora – metoda niejawna

Do rozwiązania równania transportu uzywamy metody Cranka-Nicolsona. Jest to metoda niejawna która do wyliczenia wartości w kroku następnym wykorzystuje informacje z kroku obecnego oraz poprzedniego.

3.1 Implementacja

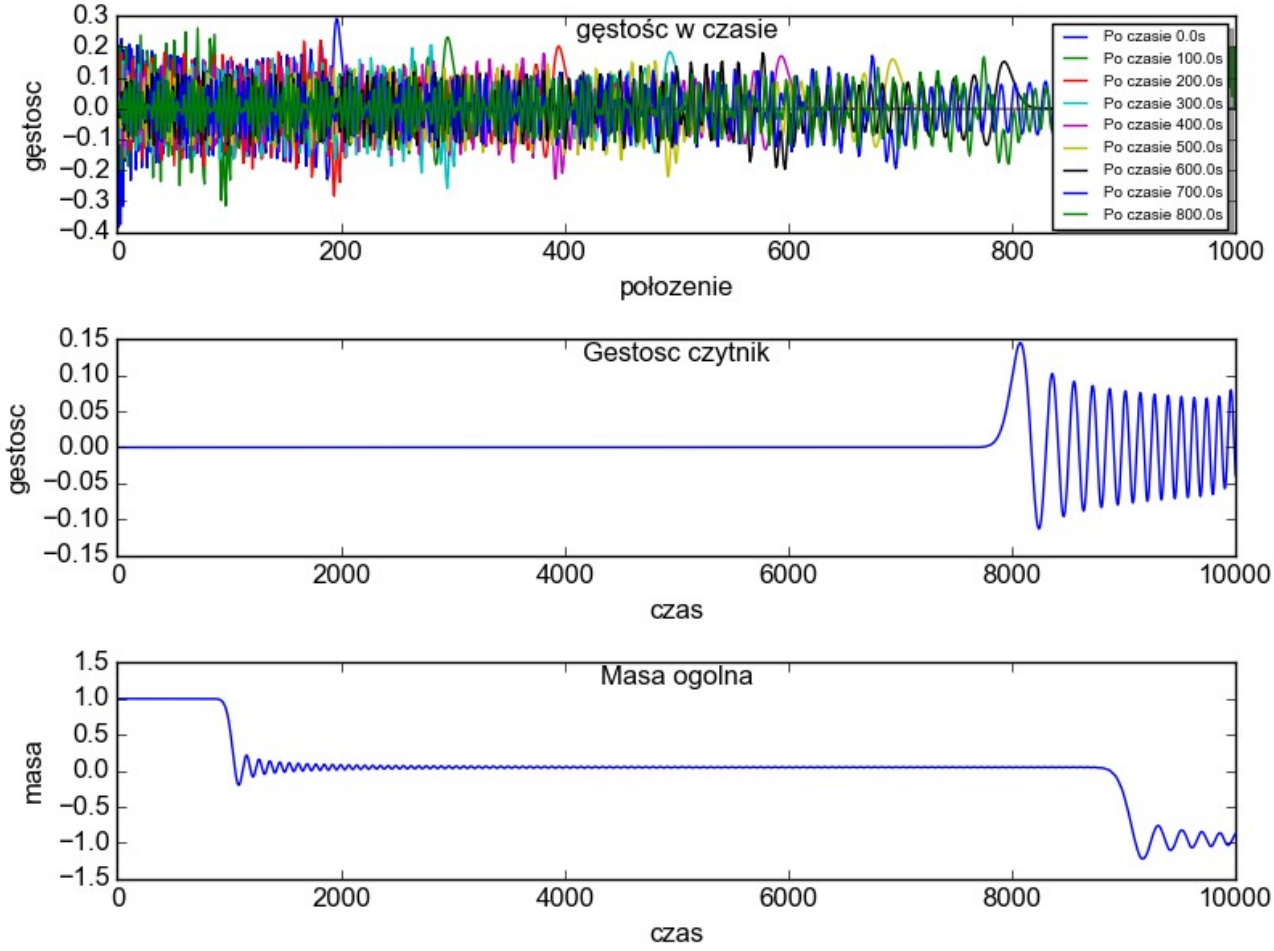
```
import numpy as np
import matplotlib
# iteracja
def iteruj(dane, AB):
    dane = np.dot(AB, dane)
    return dane
#Ustawienie wartości początkowych
matplotlib.rc('font', family='Arial')
np.set_printoptions(precision=3)
P.set_printoptions(precision=3)
dlugos = 100
szerokosc = 5
glebokosc = 1
dx = 0.1
dt = 0.1
x = int(dlugos / dx)
c = P.zeros(x)
c[:] = 0.0
i = int(10 / dx)
d = int(90 / dx)
c[i] = 1.0 / (dx * szerokosc * glebokosc)
n = int(100 / dt)
Ca = 0.1 * dt / dx
Cd = 0.01 * dt / (dx ** 2.0)
z = []
ro = []
AA = np.zeros([x, n])
BB = np.zeros([x, n])
#Wypełnienie tablic AA i BB
for i in range(1, n - 1):
    AA[i, i] = 1.0 + Cd
    AA[i, i - 1] = -Cd * 0.5 - Ca * 0.25
    AA[i - 1, i] = -Cd * 0.5 + Ca * 0.25
    BB[i, i] = 1.0 - Cd
    BB[i, i - 1] = Cd * 0.5 + Ca * 0.25
    BB[i - 1, i] = Cd * 0.5 - Ca * 0.25
AA[0, 0], AA[-1, -1] = [1.0 + Cd, 1.0 + Cd]
AA[-1, -2] = -Cd * 0.5 - Ca * 0.25
AA[-2, -1] = -Cd * 0.5 + Ca * 0.25
BB[0, 0], BB[-1, -1] = [1.0 - Cd, 1.0 - Cd]
BB[-1, -2] = Cd * 0.5 + Ca * 0.25
BB[-2, -1] = Cd * 0.5 - Ca * 0.25
AA = np.linalg.inv(AA)
AB = np.dot(AA, BB)
f, axarr = P.subplots(3, 1)
#Główna pętla
subResults = []
for x in range(10000):
    c = iteruj(c, AB)
    z.append(c[d])
    ro.append(P.sum(c[:] * (dx * szerokosc * glebokosc)))
    if x % 1000 == 0:
        subResults.append(np.copy(c))
z = np.array(z)
ro = np.array(ro)
#prezentacja wyników
for i, x in enumerate(subResults[1:]):
    axarr[0].plot(x, label="Po czasie {0}s".format(i * 1000 * dt))
axarr[0].text(.5, .9, u'gęstośc w czasie', horizontalalignment='center',
transform=axarr[0].transAxes)
axarr[0].set_xlabel(u'położenie')
axarr[0].set_ylabel(u'gęstosc')
axarr[0].legend(loc='upper left', shadow=True, prop={'size': 7})
axarr[1].plot(z)
axarr[1].text(.5, .9, u'Gestosc czytnik', horizontalalignment='center',
transform=axarr[1].transAxes)
axarr[1].set_xlabel('czas')
axarr[1].set_ylabel('gestosc')
axarr[2].plot(ro)
axarr[2].text(.5, .9, u'Masa ogolna', horizontalalignment='center',
transform=axarr[2].transAxes)
axarr[2].set_xlabel('czas')
axarr[2].set_ylabel('masa')
f.tight_layout(pad=0.4, w_pad=1.0)
```

3.2 Wynik Metody

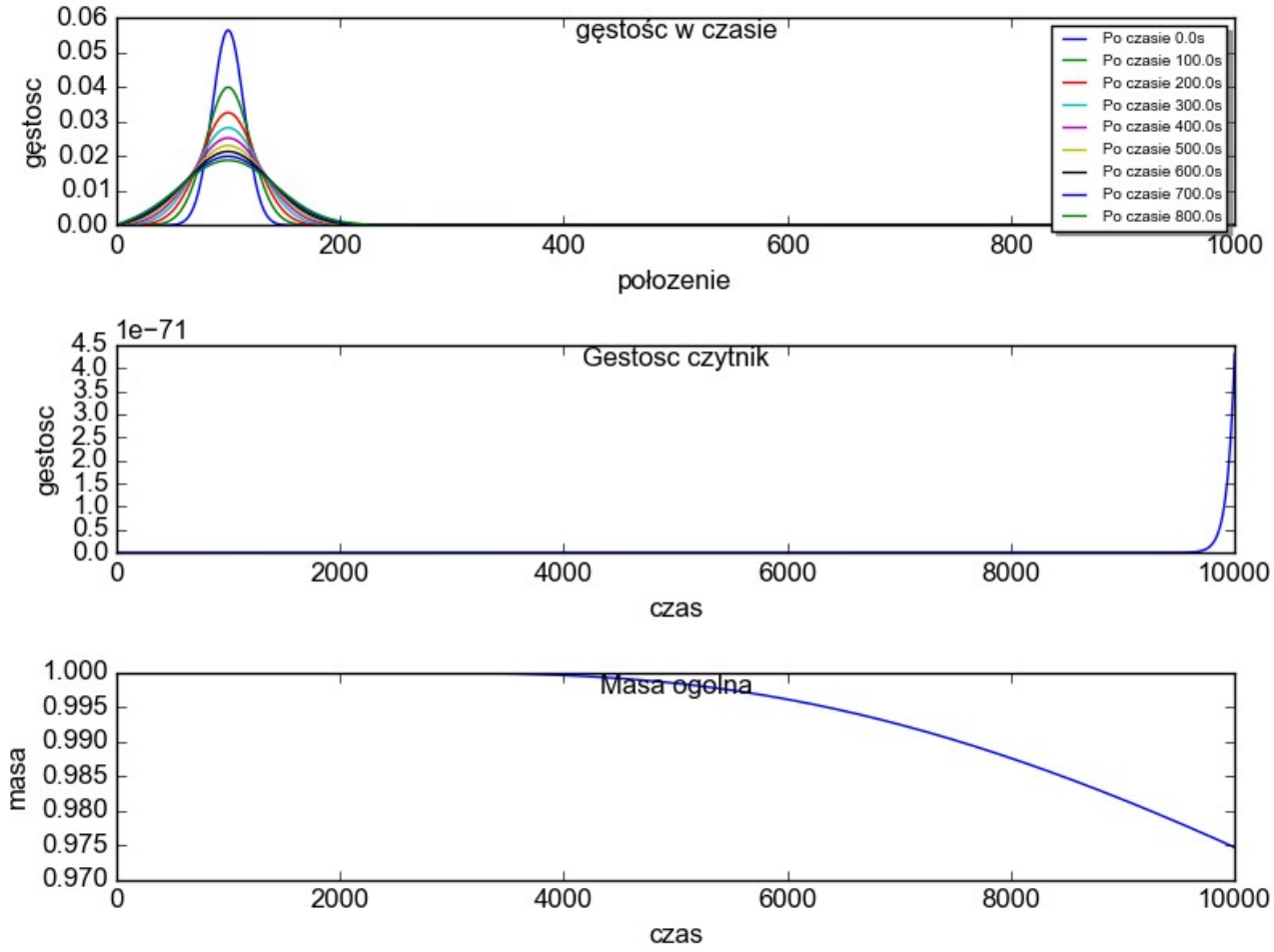


Ilustracja 5: Wyniki Metody

3.3 Wriant z samą adwekcją

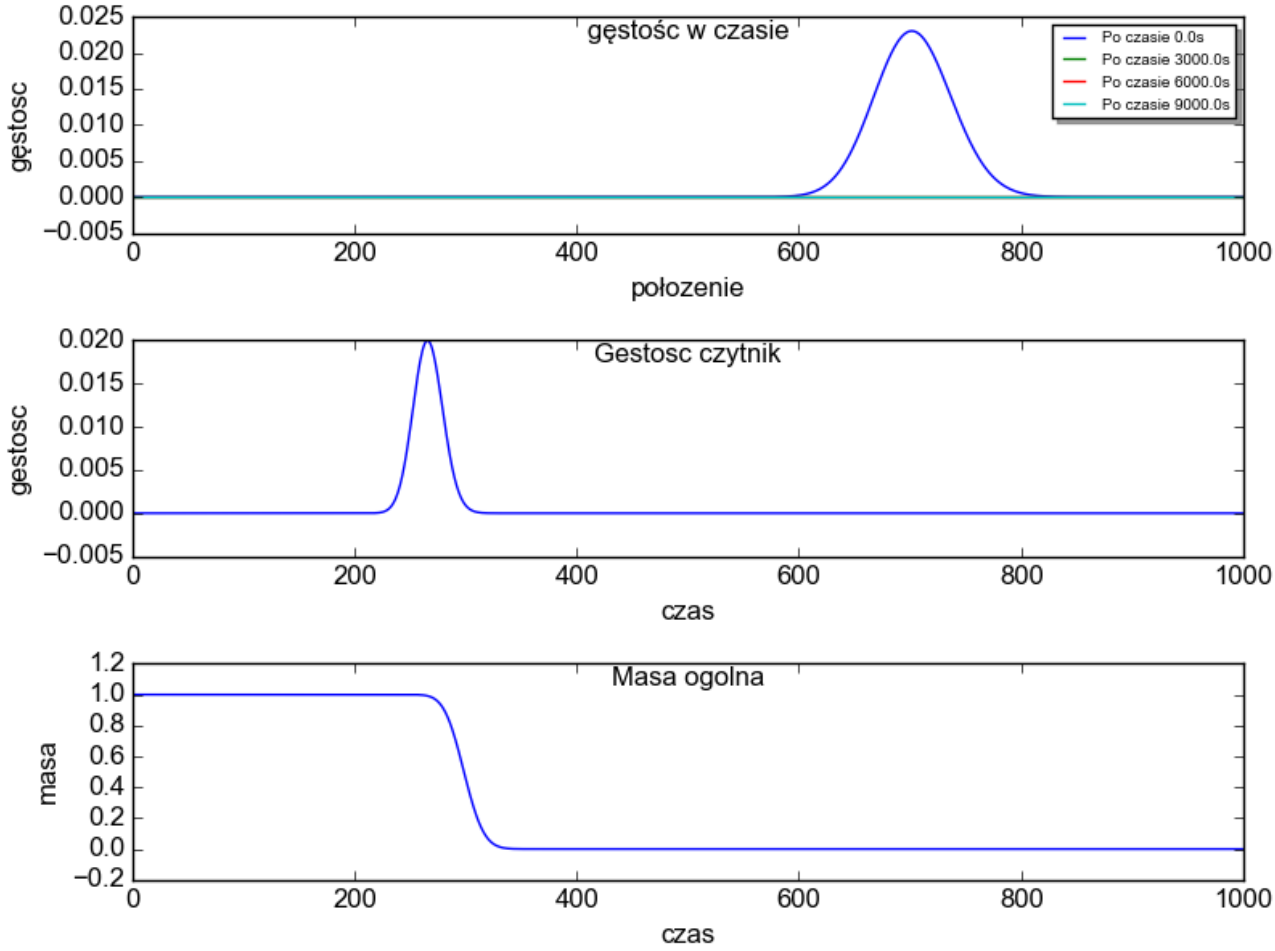


3.4 Wariant z samą dyspersją



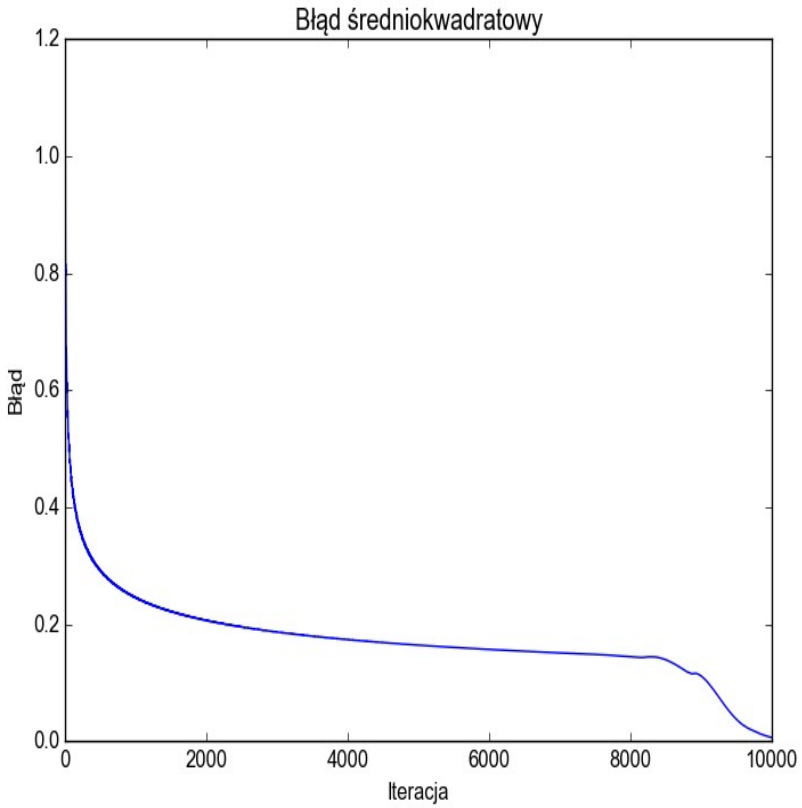
Ilustracja 6: Wariant z samą adwekcją

3.5 Graniczny krok czasowy



Ilustracja 7: Wynik dla dt=3.0s

4) Prównanie Metod



Ilustracja 8: Wykres rożnicy między metodami

5) Wnioski

Przy dobrze dobranych parametrach obie metody dają porównywalne wyniki.Jednak metoda niejawna daje dobre wyniki dla większych krokuków czsowych. Gdy w metodzie jawnej zwiększyliśmy krok czasowy do dt = 0.8 wtedy rozwiązanie równania stawało się niestabilnie numeryczne. Obie metody rożnia sie od siebie w pierwszych iteracjach jednak z czasem staja się zbieżne. Obie metody dobrze radziły sobie z modelem w którym występowała sama dyspersja ale metoda niejawna miała problemy z stabilnością dla równania w którym występowała sama adwekcja.