Fale z tarciem (waves with friction)

Wstęp do modelowania komputerowego

Adrianna Saribekyan, Ewelina Kowal, Maciej Kucharski

18 stycznia 2022

Spis treści

1 Wiadomości wstępne

1.1 Wstęp teoretyczny

1.1.1 Fale z oporem

W rzeczywistych warunkach drgania nie trwają wiecznie, ponieważ w prawdziwym świecie występuje zjawisko oporu. Wyobraźmy sobie strune drgającą w lepkim ośrodku takim jak powietrze. W przybliżeniu możemy założyć, że siła oporu jest skierowana w strone przeciwną do pionowej prędkości tej struny i jest ona proporcjonalna do wartości szybkości oraz długości struny:

$$F_f \simeq -2\kappa\Delta x\frac{\partial y}{\partial t} \tag{1}$$

gdzie κ to stała proporcjonalna do lepkości ośrodka w którym struna wibruje.

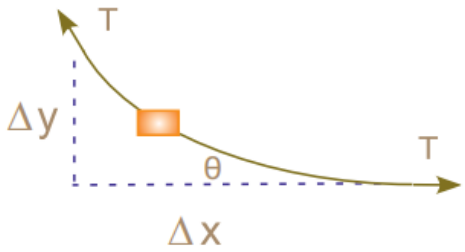
Wprowadzając wyliczoną siłę oporu (1) do równania ruchu otrzymujemy nowe równanie fali:

$$\frac{\partial^2 y}{\partial t^2} = c^2\frac{\partial^2 y}{\partial x^2} - \frac{2\kappa}{\rho}\frac{\partial y}{\partial t} \tag{2}$$

gdzie c to szybkość rozchodzenia się zaburzenia wzdłuż fali, a ρ to gęstość ośrodka.

1.1.2 Fale dla zmiennych napięć i gęstości

Fale poruszają się wolniej w rejonach o dużej gęstości i szybciej w rejonach o wysokim napięciu. Jeśli gęstość wzrasta, rośnie również napięcie, aby przyspieszyć większą masę. $T \neq const$



Rysunek 1: Różnicowy element struny pokazujący, w jaki sposób przemieszczenie struny prowadzi do siły przywracającej.

$$\tfrac{\partial}{\partial x} = [T(x)\tfrac{\partial y(x,t)}{\partial x}]\delta x = p(x)\delta x\tfrac{\partial^2 u(x,t)}{\partial t^2},$$

$$\tfrac{\partial T(x)}{\partial x}\tfrac{\partial y(x,t)}{\partial x} + T(x)\tfrac{\partial^2 y(x,t)}{\partial x^2} = p(x)\tfrac{\partial^2 y(x,t)}{\partial t^2}$$

Załóżmy, że gęstość i naprężenia są proporcjonalne:

$$p(x) = p_0 e^{\alpha x}, T(x) = T_0 e^{\alpha x} \tag{3}$$

Proporjonalność jednak okazuje się tutaj jedynie przybliżeniem. Po podstawieniu do równania 4 otrzymujemy:

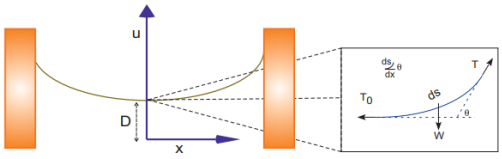
$$\frac{\partial^2 y(x,t)}{\partial x^2} + \alpha\frac{\partial y(x,t)}{\partial x} = \frac{1}{c^2}\frac{\partial^2 y(x,t)}{\partial t^2}, c^2 = \frac{T_0}{p_0} \tag{4}$$

Odpowiednie równanie różniczkowe wynika z zastosowania przybliżeń różnic centralnych dla pochodnych:

$$y_{i,j+1} = 2y_{i,j} - y_{i,j-1}\frac{\alpha c^2(\Delta t)^2}{2\Delta x}[y_{i+1,j} - y_{i,j}] + \frac{c^2}{c'^2}[y_{i+1,j} - y_{i,j} - 2yi, j] \tag{5}$$

$$y_{i,2} = y_{i,1} + \frac{c^2}{c'^2}[y_{i+1,1} + y_{i-1,1} - 2y_{i,1}] + \frac{\alpha c^2 (\Delta t)^2}{2\Delta x}[y_{i+1,1} - y_{i,1}] \tag{6}$$

Do tego momentu ignorowaliśmy wpływ grawitacji na kształt i napięcie naszej struny, co sprawdza się gdy struna posiada nieduże ugięcie. Jeśli jednak struna jest masywna, powiedzmy, jak łańcuch lub ciężki kabel, ugięcie w środku spowodowane grawitacją może być dość duże (jak na rysunku poniżej), a wynikające z tego zmiany kształtu i naprężenia muszą zostać uwzględnione w równaniu fali.



Rysunek 2: Jednolita struna i wykres rozkładania sił.

Ponieważ naprężenie nie jest już jednolite, fale przemieszczają się szybciej w pobliżu końców struny, które są bardziej naprężone, ponieważ muszą utrzymać cały ciężar struny.

1.2 Pochodna kształtu krzywej łańcuchowej

Rozważmy strune o jednolitej gęstości ρ na którą działa grawitacja. Najpierw musimy rozwiązać problem w stanie statycznym i określić kształt $u(x)$ (kształt struny w stanie równowagi) oraz napięcie $T(x)$. Na rysunku (Rys.2) widać, że fragment o długości s i masie W jest równoważony przez pionową składową napięcia T. Napięcie pionowe T_0 jest natomiast równoważone przez poziomą składową T:

$$T(x)sin\Theta = W = \rho g s \wedge T(x)cos\Theta = T_0 \rightarrow tan\Theta = \frac{\rho g s}{T_0} \tag{7}$$

Przekształcamy równanie (7), zamieniając $tan\Theta$ na pochodną $\frac{du}{dx}$ i różniczkujemy ze względu na x:

$$\frac{du}{dx} = \frac{\rho g}{T_0} s \rightarrow \frac{d^2u}{dx^2} = \frac{\rho g}{T_0}\frac{ds}{dx} \tag{8}$$

Podstawiając za $ds = \sqrt{dx^2 + du^2}$ otrzymujemy nasze równanie różniczkowe:

$$\frac{d^2u}{dx^2} = \frac{1}{D}\frac{\sqrt{dx^2 + du^2}}{dx} = \frac{1}{D}\sqrt{1 + (\frac{du}{dx})^2} \tag{9}$$

gdzie $D = \frac{T_0}{\rho g}$ to stała wyrażana przez jednostkę długości.

Możemy zauważyć, że równanie (9) jest równaniem krzywej łańcuchowej i jego rozwiązaniem jest:

$$u(x) = Dcosh\frac{x}{D} \tag{10}$$

Rysunek (Rys.2) ustaliliśmy tak, że $x = 0$ leży w centrum struny, $y = D$ oraz $T = T_0$. Z równania (10) wynika, że $s = D\frac{du}{dx}$.

Podstawiając wzór (10) do tego właśnie wzoru na s otrzymujemy równanie dla $s(x)$ oraz następnie (za pomocą równania (7)) dla $T(x)$:

$$s(x) = Dsinh\frac{x}{D} \rightarrow T(x) = T_0\frac{ds}{dx} = \rho g u(x) = T_0 cosh\frac{x}{D} \tag{11}$$

To właśnie ta obliczona przez nas zmiana w napięciu powoduje zmianę prędkości fali dla różnych pozycji na strunie.

2 Implementacja algorytmów w Pyhonie

2.1 Rozwiązanie równania falowego dla przypadku z tarciem

2.1.1 Rozwiązanie przy użyciu biblioteki *vpython*

Pierwszy kod przedstawiający rozwiązanie równania falowego dla przypadku z traciem rozpoczęliśmy od zaimportowania potrzebnych bibliotek. *Vpython* pozwala tworzyć grafiki 3D oraz wywoływać je w oknie. Użyte zostały również *IPython.display* oraz *numpy*.

Kolejnym krokiem było określenie atrybutów graficznych - okna wyświetlania (wiersz 5), *vibst*, czyli struny (vibrating) oraz kulek, w tym ich pozycji (6-12).

```
1  from vpython import *
2  from IPython.display import display
3  import numpy as np
4
5  g = display(width = 600, height = 300, title = 'Vibrating string')
6  vibst = curve(x = list(range(0, 100)), y = list(range(0, 100)), color = color.yellow)
7
8  ball1 = sphere(pos = vector(100,0, 0), color = color.red, radius = 2)
9  ball2 = sphere(pos = vector(-100,0, 0), color = color.red, radius = 2)
10 ball1.pos
11 ball2.pos
12 vibst.radius = 1.0
13
```

Rysunek 3: Wiersze 1-13 programu rozwiązującego równanie fali z tarciem przy użyciu biblioteku *vpython*.

Następnie zostały zdefiniowane stałe zgodnie z informacjami w treści zadania (14-21). W wierszu 23 została stworzona tablica zer odpowiadająca ilości x-ów oraz t. Następnie przeszliśmy do trzech pętli for, gdzie tablica zostałą zapełniona dla początkowych wartości czasu (24-30). Następnie w oknie zostaje narysowana struna (31).

```
14 rho = 0.01  # string density
15 ten = 40.  # string tension
16 c = sqrt(ten/rho)  # Propagation speed
17 cl = c  # CFL criterium
18 ratio = c*c/(cl*cl)
19 k = 30  # kappa
20 ro = 10  # density
21 delta_t = 0.0001  # dt
22
23 xi = np.zeros((101,3), float)  # 101 x's & 3 t's
24 for i in range (0, 81):
25     xi[i, 0] = 0.00125*i;
26 for i in range(81, 101):
27     xi[i, 0] = 0.1 - 0.005*(i - 80)  #IC
28 for i in range (0, 100):  # 1st t step
29     vibst.x[i] = 2.0*i - 100.0  #assign, scale x
30     vibst.y[i] = 300.*xi[i, 0]  #assign, scale y
31 vibst.pos # draw string
```

Rysunek 4: Wiersze 14-31 programu rozwiązującego równanie fali z tarciem przy użyciu biblioteki *vpython*.

W wierszu 34 przeszliśmy do zapisania wartości w tablicy dla kolejnych (późniejszych) wartości czasu. W wierszach 35 oraz 39 widzimy odpowiednio przedstawione równanie falowe z elementem zawierającym tarcie.

```
33  # Later time steps
34  for i in range(1, 100):
35      xi[i, 1] = xi[i, 0] + 0.5*ratio*(xi[i+1, 0] + xi[i-1, 0] - 2*xi[i, 0]) - 0.5 * k / ro * (xi[i, 2]-xi[i, 0]) / delta_t
36  while 1:  #continue plotting till user quits
37      rate(50)  #delays plotting, (bigger = slower)
38      for i in range(1, 100):
39          xi[i, 2] = 2.*xi[i, 1] - xi[i, 0] + ratio * (xi[i+1, 1]+ xi[i-1, 1] - 2*xi[i, 1]) - 0.5 * k / ro * (xi[i, 2]-xi[i, 0]) / delta_t
40      for i in range (1, 100):
41          vibst.x[i] = 2.*i - 100.0  # scaled x
42          vibst.y[i] = 300.*xi[i, 2]  # scaled y
43      vibst   # vibst.pos
44      for i in range(0, 101):
45          xi[i, 0] = xi[i, 1]  #recycle array
46          xi[i, 1] = xi[i, 2]
47
48  print("Done!")
```

Rysunek 5: Wiersze 33-48 programu rozwiązującego równanie fali z tarciem przy użyciu biblioteki *vpython*.

Wydruk naszego rozwiązania w oknie wywołanym dzięki funkcji *vpython* wygląda następująco.

Rysunek 6: Działanie programu rozwiązującego równanie fali z tarciem przy użyciu biblioteki *vpython*.

W związku z problemem wbudowanej w bibliotekę *vpython* funkcji *pos*, nie została narysowana struna, a jedynie kulki (węzły).

```
Traceback (most recent call last):
  File "C:\Users\ANI\PycharmProjects\modelowanieprojekt2\venv\wave.py", line 31, in <module>
    vibst.pos # draw string
  File "C:\Users\ANI\PycharmProjects\modelowanieprojekt2\venv\lib\site-packages\vpython\vpython.py", line 1966, in pos
    raise AttributeError('object does not have a "pos" attribute')
AttributeError: object does not have a "pos" attribute
```

Rysunek 7: Błąd programu rozwiązującego równanie fali z tarciem przy użyciu biblioteki *vpython*.

2.1.2 Rozwiązanie przy użyciu biblioteki *matplotlib*

Drugi kod przedstawiający rozwiązanie równania falowego dla przypadku z traciem rozpoczęliśmy od zaimportowania potrzebnych bibliotek. *matplotlib.animation* pozwala tworzyć animacje. Użyte zostały również *matpplotlib.pyplot* oraz *numpy*.

W wierszach 5-14 określiliśmy wartości stałych.

```
1   from numpy import *
2   import numpy as np, matplotlib.pyplot as plt
3   import matplotlib.animation as animation
4
5   rho = 0.01;
6   ten = 40.;
7   c = sqrt(ten / rho)
8   cl = c;
9   ratio = c * c / (cl * cl)
10  xi = np.zeros((101, 3), float)
11  k = range(0, 101)
12  dt = 0.0001
13  kappa = 30
14  ro = 10
15
16
```

Rysunek 8: Wiersze 1-16 programu rozwiązującego równanie fali z tarciem przy użyciu biblioteki *matplotlib*.

Następnie została stworzona funkcja, gdzie znajdują się tablice z określonymi warościami dla początkowych etapów czasu (17-19). Kolejno zdefiniowano funkcję *animate*, gdzie w pętli widzimy równanie falowe zawierające dodatkowo składnik odpowiadający za tarcie (22-29).

```
17  def Initialize():
18      for i in range(0, 81): xi[i, 0] = 0.00125 * i
19      for i in range(81, 101):  xi[i, 0] = 0.1 - 0.005 * (i - 80)
20
21
22  def animate(num):
23      for i in range(1, 100):
24          xi[i, 2] = 2. * xi[i, 1] - xi[i, 0] + ratio * (xi[i + 1, 1] + xi[i - 1, 1] - 2 * xi[i, 1]) - 0.5 * kappa / ro * (xi[i, 2]-xi[i, 0]) / dt
25          line.set_data(k, xi[k, 2])
26          for m in range(0, 101):
27              xi[m, 0] = xi[m, 1]
28              xi[m, 1] = xi[m, 2]
29              return line
30
31
```

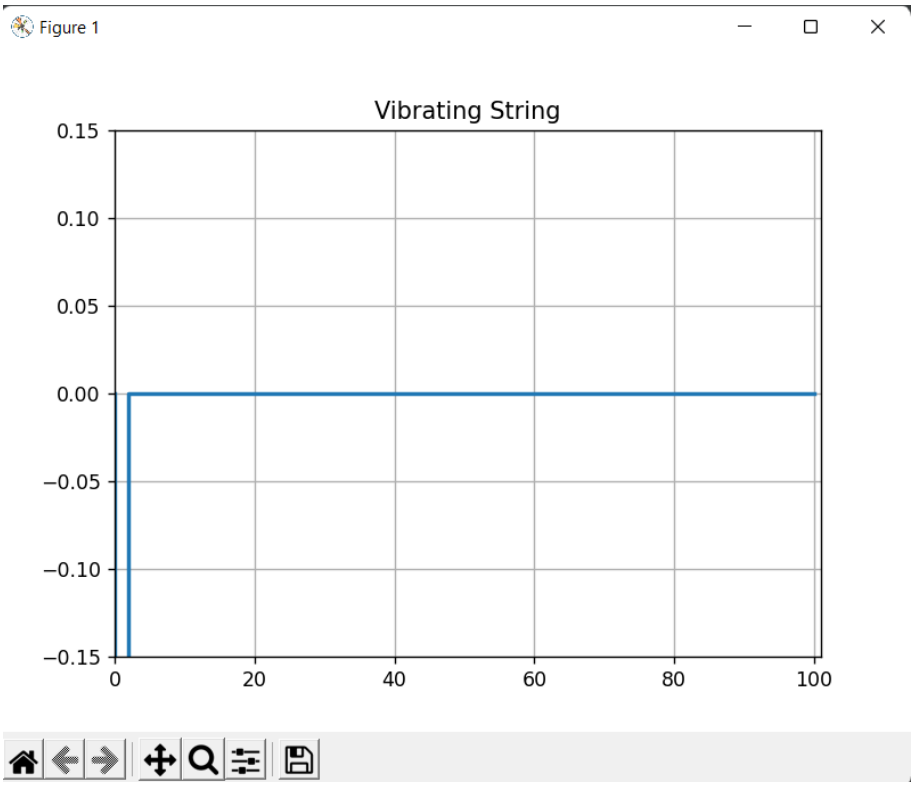
Rysunek 9: Wiersze 17-31 programu rozwiązującego równanie fali z tarciem przy użyciu biblioteki *matplotlib*.

W wierszy 32 zostaje wywołana funkcja *Initialize*. Kolejnymi krokami są graficzne ustawienia oraz pętla gdzie liczone są kolejne wartości równania falowego z tarciem. Na końcu widzimy użycie animacji na zmiennej przechowującej wykres oraz wyświetlanie końcowego wykresu.

```
32  Initialize()
33  fig = plt.figure()
34  ax = fig.add_subplot(111, autoscale_on=False, xlim=(0, 101), ylim=(-0.15, 0.15))
35  ax.grid()
36  plt.title("Vibrating String")
37  line, = ax.plot(k, xi[k, 0], lw=2)
38  for i in range(1, 100):
39      xi[i, 1] = xi[i, 0] + 0.5 * ratio * (xi[i + 1, 0] + xi[i - 1, 0] - 2 * xi[i, 0])- 0.5 * kappa / ro * (xi[i, 2]-xi[i, 0]) / dt
40
41  anim = animation.FuncAnimation(fig, animate, 1)
42  plt.show()
43  print("finished")
```

Rysunek 10: Wiersze 32 -43 programu rozwiązującego równanie fali z tarciem przy użyciu biblioteki *matplotlib*.

Wynikiem tego programu jest poniższy wykres. W związku z nieprawidłowym działaniem funkcji odpowiadającej za animacje, dostajemy jedynie wykres.



Rysunek 11: Wynik programu rozwiązującego równanie fali z tarciem przy użyciu biblioteki *matplotlib*.

2.2 Rozwiązanie równania falowego na krzywej łańcuchowej z tarciem

Trzeci kod przedstawiający rozwiązanie równania falowego na krzywej łańcuchowej z tarciem rozpoczęliśmy od zaimportowania potrzebnych bibliotek. *matplotlib.pyplot* oraz *numpy*.

W wierszach 4-10 określiliśmy wartości stałych. Następnie stworzyliśmy puste tablice, które posłużą do wygenerowania wykresu (12-13) oraz tablicę składającą się z zer (15) i otworzyliśmy do zapisu danych dwa pliki .txt. Jeśli nie ma takich plików, zostaną stworzone (16).

```
1   from numpy import *
2   import matplotlib.pyplot as plt
3
4   dt = 0.0001
5   dx = 0.01
6   T = 40. #tension
7   rho = 0.1  # density
8   maxtime = 100
9   kappa = 30
10  D = T/(rho*9.8)
11
12  xx = []
13  yy = []
14
15  x = zeros ((512,3), float)  # (x,t)
16  q = open('CatFrict.dat', 'w'); rr=open('Catfunct.dat', 'w+t')
17
```

Rysunek 12: Wiersze 1-17 programu rozwiązującego równanie fali na krzywej łańcuchowej z tarciem.

Następnie zostały stworzone funkcje zapisujące do tablic wartości dla początkowych kroków czasu t.

```
18  for i in range (0,101):
19      x[i][0] = -0.08*sin(pi*i*dx)
20
21  for i in range (1,100):  # First step
22      x[i][1]=(dt*(T/rho)*((x[i+1][0]-x[i][0])
23      /dx*(exp((i-50)*dx/D)
24      -exp(-(i-50)*dx/D))/D + (exp((i-50)*dx/D)
25      +exp(-(i-50)*dx/D))*(x[i+1][0]+x[i-0][0]
26      -2.0*x[i][0])/(pow(dx,2)))
27      -2*kappa*x[i][0]+2*x[i][0]/dt)/ (2*kappa+(2./dt))
28
```

Rysunek 13: Wiersze 18-28 programu rozwiązującego równanie fali na krzywej łańcuchowej z tarciem.

Natępnie została stworzona pętla, gdzie do tablic zostają wpisane wartości dla kolejnych kroków czasu. Widzimy również, że w linijkach 44 oraz 46 do plików zostają wpisane wartości. Natomiast w wierszach 48 i 49 dodajemy do tablic kolejne wartości.

```
29  for k in range (0,300):  # next steps
30      for i in range (1,100):
31          x[i][2]=(dt*(T/rho)*((x[i+1][1]-x[i][1])
32          /dx*(exp((i-50)*dx/D)-exp(-(i-50)*dx/D))/D
33          +(exp((i-50)*dx/D)+exp(-(i-50)*dx/D))*
34          (x[i+1][1]+x[i-1][1]-2.0*x[i][1])/(pow(dx,2)))
35          -2*kappa*x[i][1]
36          -(-2*x[i][1]+x[i][0])/dt)/(2*kappa+(1./dt))
37      for i in range(1, 101):
38          x[i][0] = x[i][1]
39          x[i][1] = x[i][2]
40          if (k % 4 == 0 or k == 0):
41              for i in range(0, 100):
42                  a1 = exp((i - 50.) * dx / D)
43                  a2 = exp(-(i - 50.) * dx / D)
44                  rr.write("%7.3f" % (D * (a1 + a2)))
45                  rr.write("\n")
46                  q.write('%7.3f' % (x[i, 2]))
47                  q.write("\n")
48                  xx += [D * (a1 + a2)]
49                  yy +=[x[i, 2]]
50              q.write("\n");
51              rr.write("\n");
```

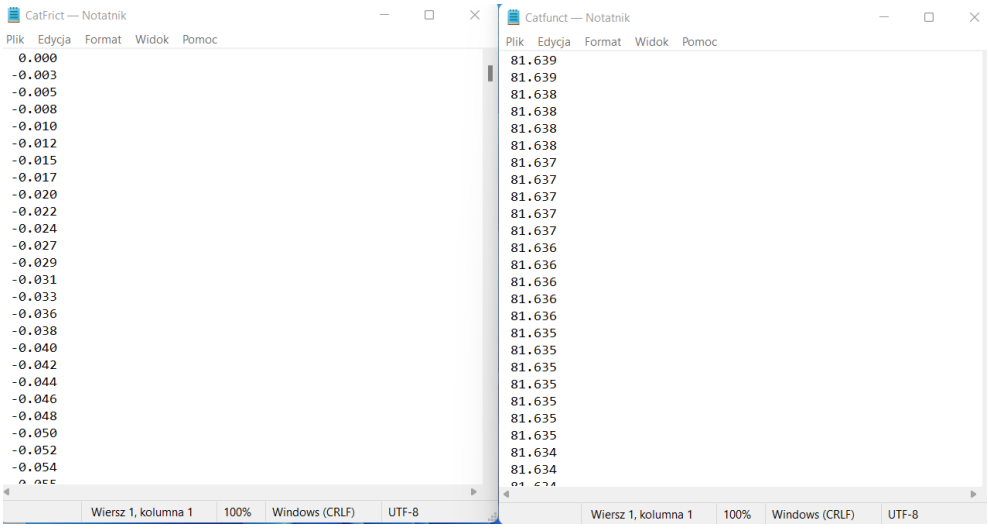
Rysunek 14: Wiersze 29-51 programu rozwiązującego równanie fali na krzywej łańcuchowej z tarciem.

W ostatnim etapie zamykamy pliki (53-54) oraz generujemy wykres (57-58).

```
52
53  rr.closed
54  q.closed
55  print("data stored in CatFrict.dat and CatFunct.dat")
56
57  plt.plot(xx,yy)
58  plt.show()
```

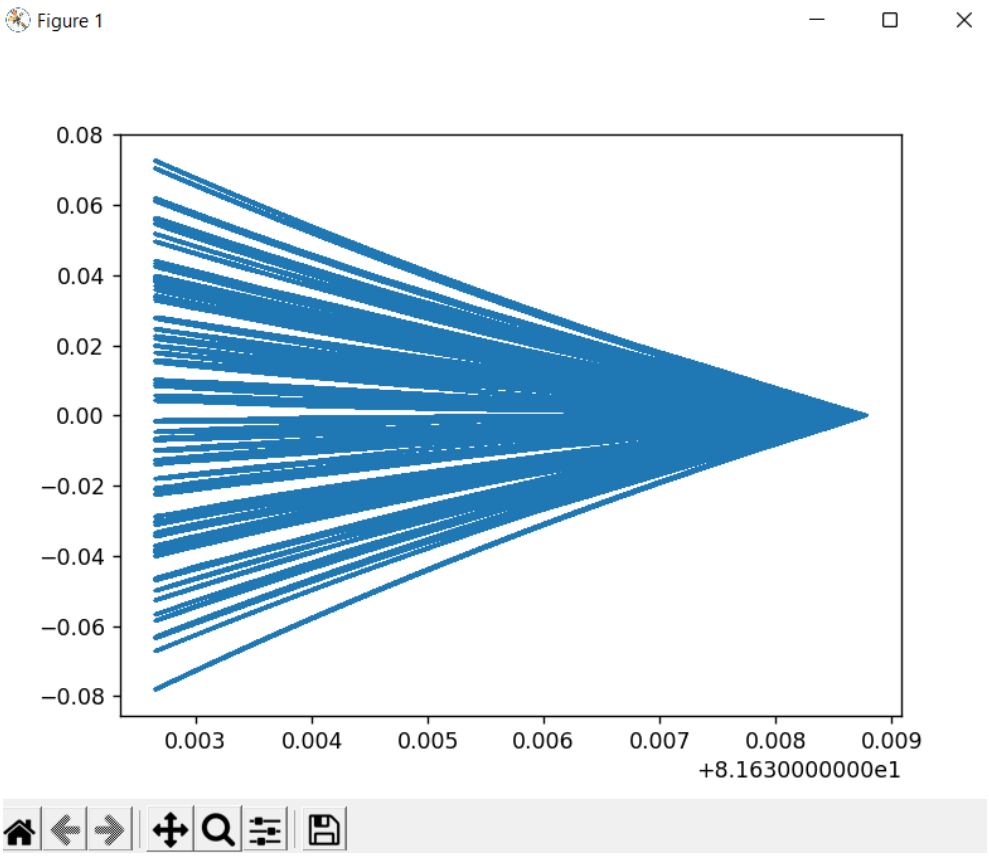
Rysunek 15: Wiersze 52-58 programu rozwiązującego równanie fali na krzywej łańcuchowej z tarciem.

Zgodnie z założeniem, w plikach znajdują się dane wyliczone w programie.



Rysunek 16: Wynik (zapisane wartości) programu rozwiązującego równanie fali na krzywej łańcuchowej z tarciem.

Wygenerowany wykres stworzony z danych wyliczonych w programie (te same dane, co w plikach .txt) wygląda jak poniżej.



Rysunek 17: Wynik (wykres) programu rozwiązującego równanie fali na krzywej łańcuchowej z tarciem.

3 Bibliografia

Literatura

[1] http://kfe.fjfi.cvut.cz/ kucharik/edu/PF/1/lit/Landau_Paez-CP_Python-2018.pdf