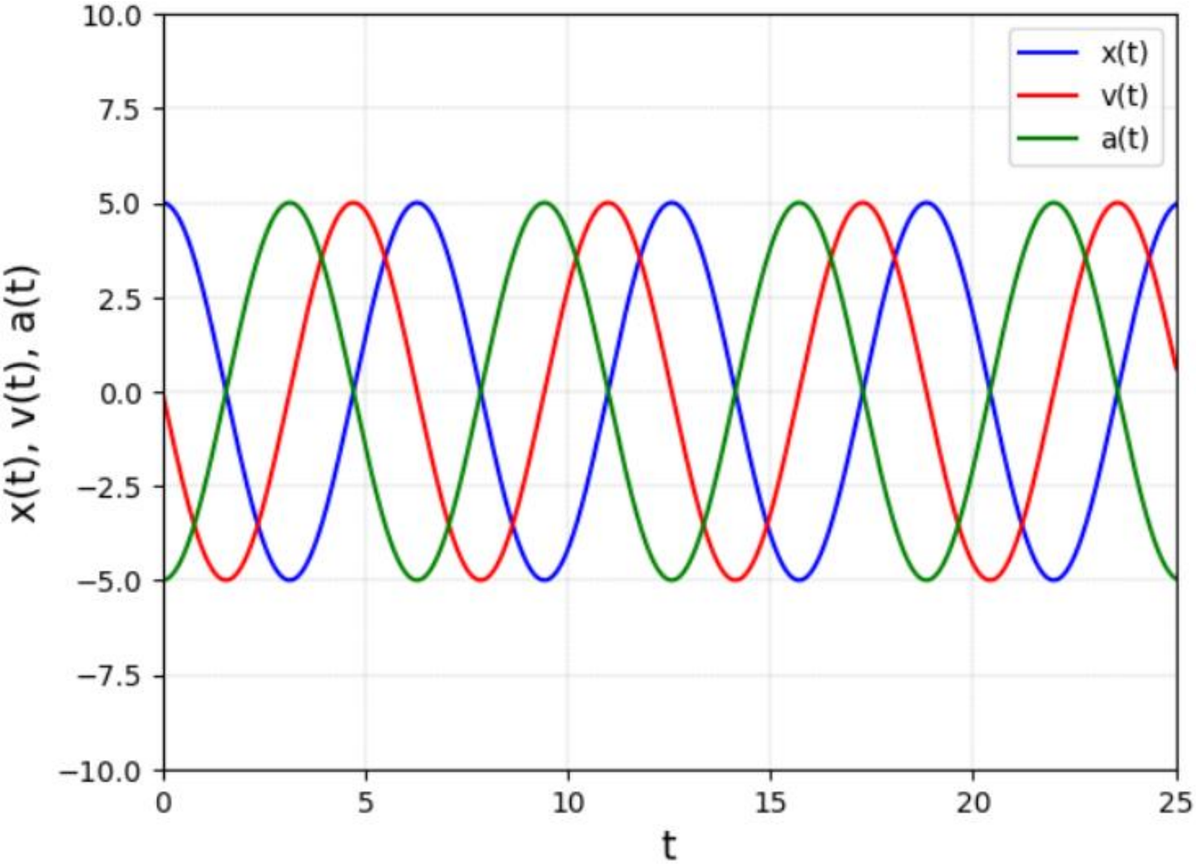
Karol Ratajczyk ETI

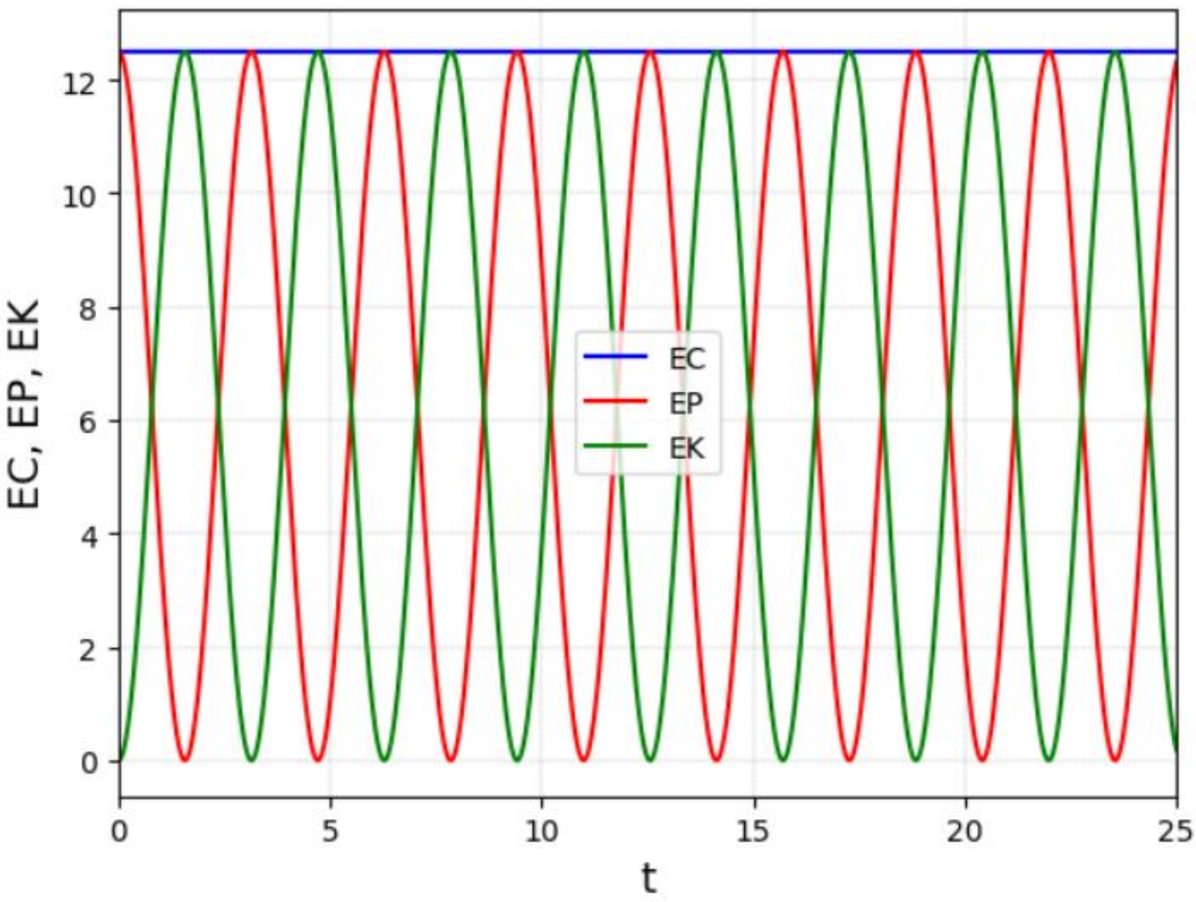
Zad 2
Rozwiązać numerycznie równanie różniczkowe **nietłumionego** oscylatora harmonicznego korzystając z algorytmów z rodziny Verleta (standard, leap-frog i prędkościowy). Wykonać wykresy x(t), v(t), a(t), wykres przestrzeni fazowej p(x), sprawdzić zachowanie energii w czasie oraz odchylenia standardowego energii całkowitej od kroku czasowego (w skali logarytmicznej). **Uwzględnić fakt, że ze względu na tłumienie, energia całkowita zmienia się.**

Zauważyłem że w ostatnim zdaniu się coś nie zgadza, wiec uznałem że wykonam zadanie bez brania pod uwagi ostatniego zdania

```
import math
import numpy as np
import matplotlib.pyplot as plt
A = 5
k = 1
m = 1
f = 1
omega = math.sqrt(k/m)
dt = 0.01
t = np.arange(0, 100, dt)


#Standardowy
x_s = []
x_s.append(A)
x_s.append(math.sqrt(A**2 - omega*omega*(A*dt)**2))

v_s = []
v_s.append(0)

a_s = []
a_s.append(-omega*omega*x_s[0])
a_s.append(-omega*omega*x_s[1])

for i in range(2, len(t)):
 x_s.append(2*x_s[i-1]-x_s[i-2]+(dt**2)*a_s[i-1])
 a_s.append(-omega*omega*x_s[i])
 v_s.append((x_s[i]-x_s[i-2])/(2*dt))

ostatni_x = 2*x_s[i-1]-x_s[i-2]+(dt**2)*a_s[i-1]
v_s.append((ostatni_x-x_s[i-2])/(2*dt))

EK_s = []
```

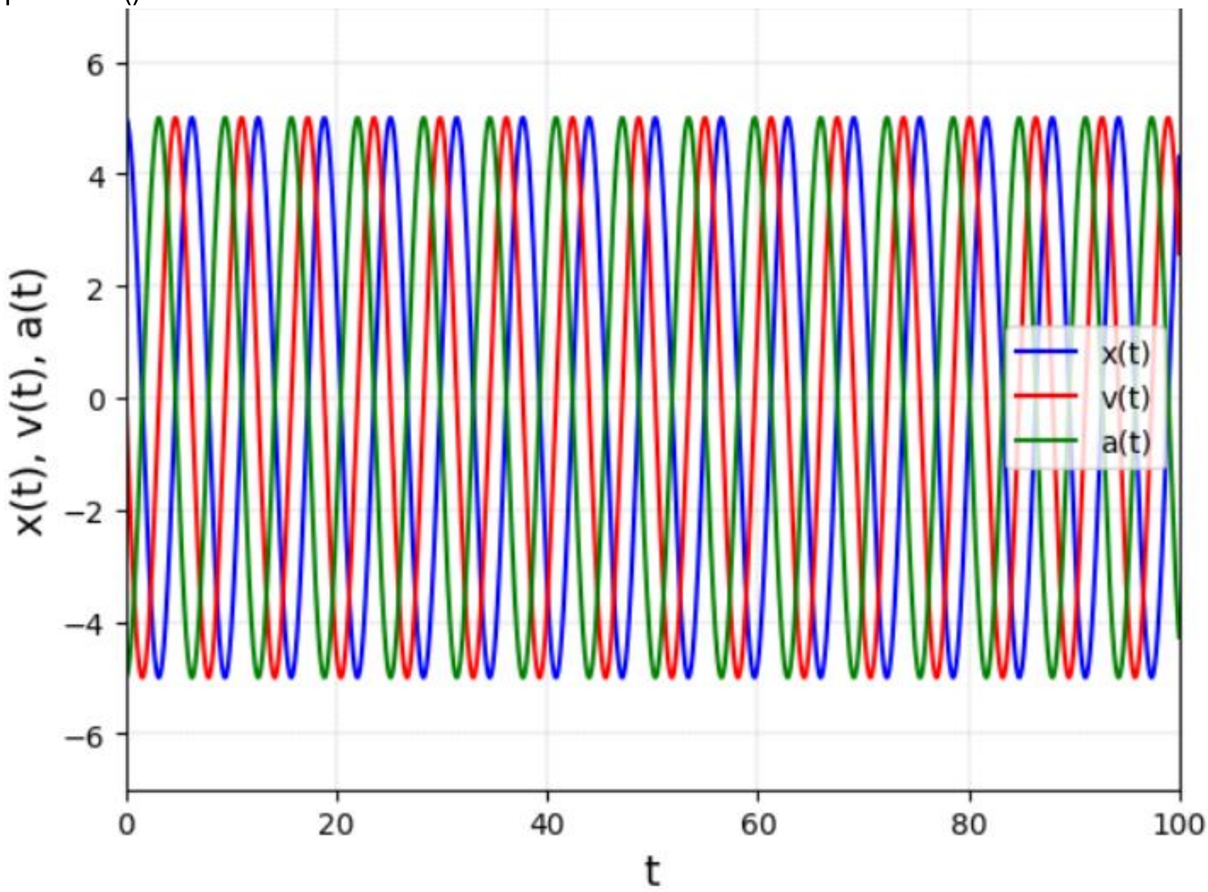
```
EP_s = []
EC_s = []

#energia
for i in range(len(x_s)):
 EK_s.append(0.5*m*v_s[i]**2)
 EP_s.append(0.5*k*(x_s[i])**2)
 EC_s.append(EK_s[i]+EP_s[i])

plt.plot(t, x_s, '-b', t, v_s, '-r', t, a_s, '-g')
plt.xlim(0, 100)
plt.ylim(-7, 7)
plt.xlabel('t', fontsize = '14')
plt.ylabel('x(t), v(t), a(t)', fontsize = '14')
plt.legend(['x(t)', 'v(t)', 'a(t)'])
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.2)
plt.show()

plt.plot(t, EC_s, '-b', t, EP_s, '-r', t, EK_s, '-g')
plt.xlim(0, 100)
plt.xlabel('t', fontsize = '14')
plt.ylabel('EC, EP, EK', fontsize = '14')
plt.legend(['EC', 'EP', 'EK'])
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.2)
plt.show()
```

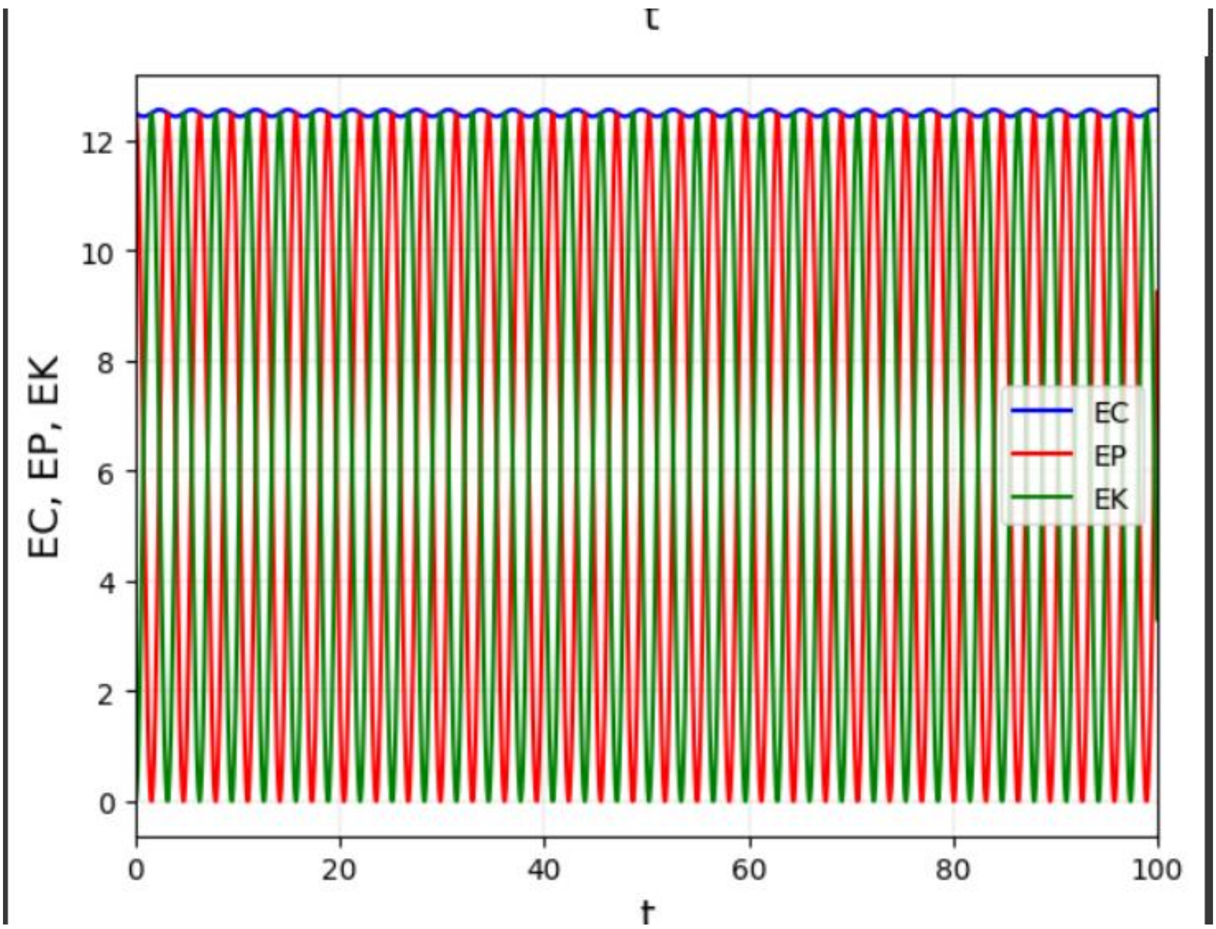

x(t)
v(t)
a(t)
x(t), v(t), a(t)
t
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
5
10
15
20
25


EC
EP
EK
EC, EP, EK
t
12
10
8
6
4
2
0
0
5
10
15
20
25

```
#leap frog

import math
import numpy as np
import matplotlib.pyplot as plt
A = 5
k = 1
m = 1
f = 1
omega = math.sqrt(k/m)
dt = 0.01
t = np.arange(0, 100, dt)


x_l = []
x_l.append(A)

v_l = []
v_l.append(0)


a_l = []
a_l.append(-omega*omega*x_l[0])

for i in range(1,len(t)):
  a_l.append(-omega*omega*x_l[i-1])
  v_l.append(v_l[i-1] + (a_l[i] * dt))
  x_l.append(x_l[i-1] + v_l[i] * dt)

plt.plot(t, x_l, '-b', t, v_l, '-r', t, a_l, '-g')
plt.xlim(0, 100)
plt.ylim(-7, 7)
plt.xlabel('t', fontsize = '14')
plt.ylabel('x(t), v(t), a(t)', fontsize = '14')
plt.legend(['x(t)', 'v(t)', 'a(t)'])
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.2)
plt.show()

EK_l = []
EP_l = []
EC_l = []
for i in range(len(x_l)):
 EK_l.append(0.5*m*v_l[i]**2)
 EP_l.append(0.5*k*(x_l[i])**2)
```

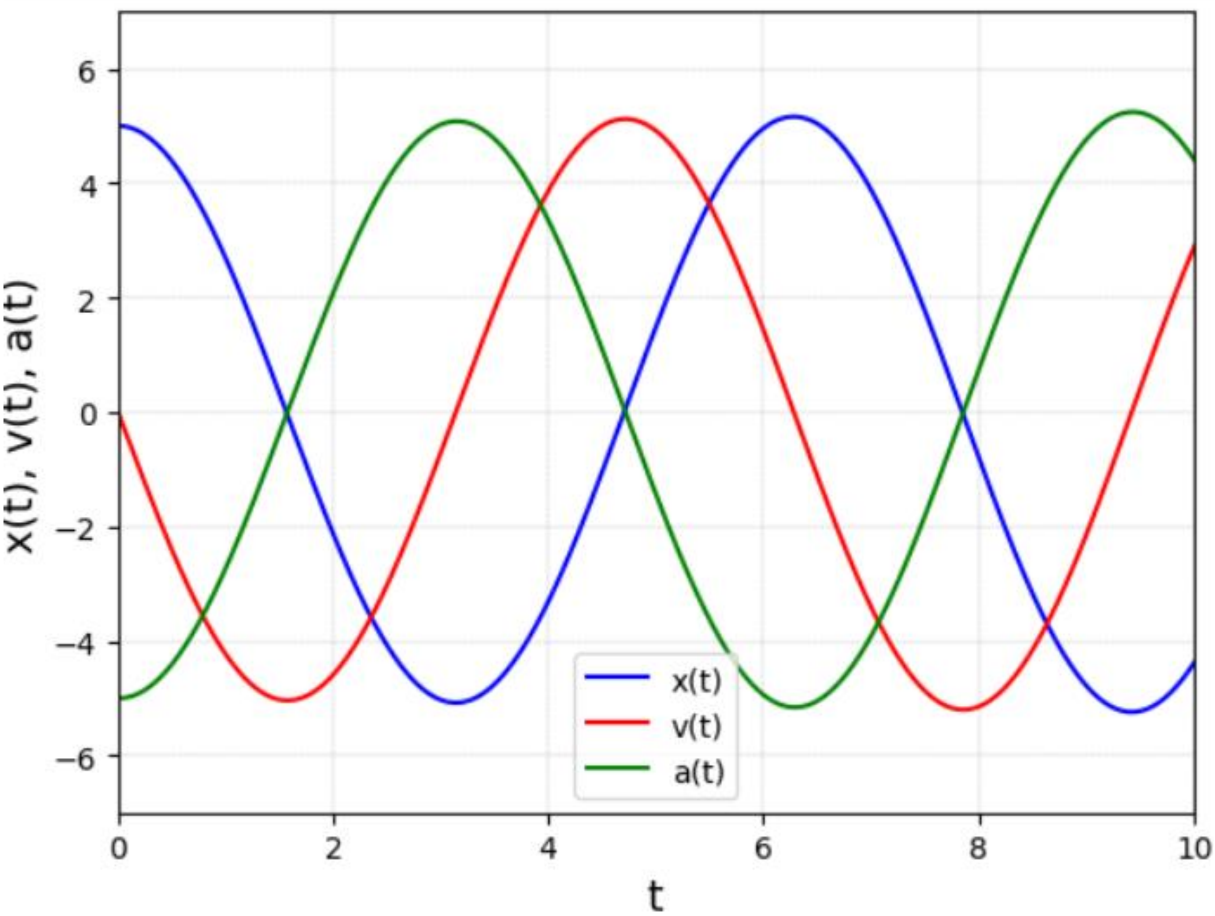
```
EC_l.append(EK_l[i]+EP_l[i])

plt.plot(t, EC_l, '-b', t, EP_l, '-r', t, EK_l, '-g')
plt.xlim(0, 100)
plt.xlabel('t', fontsize = '14')
plt.ylabel('EC, EP, EK', fontsize = '14')
plt.legend(['EC', 'EP', 'EK'])
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.2)
plt.show()
```

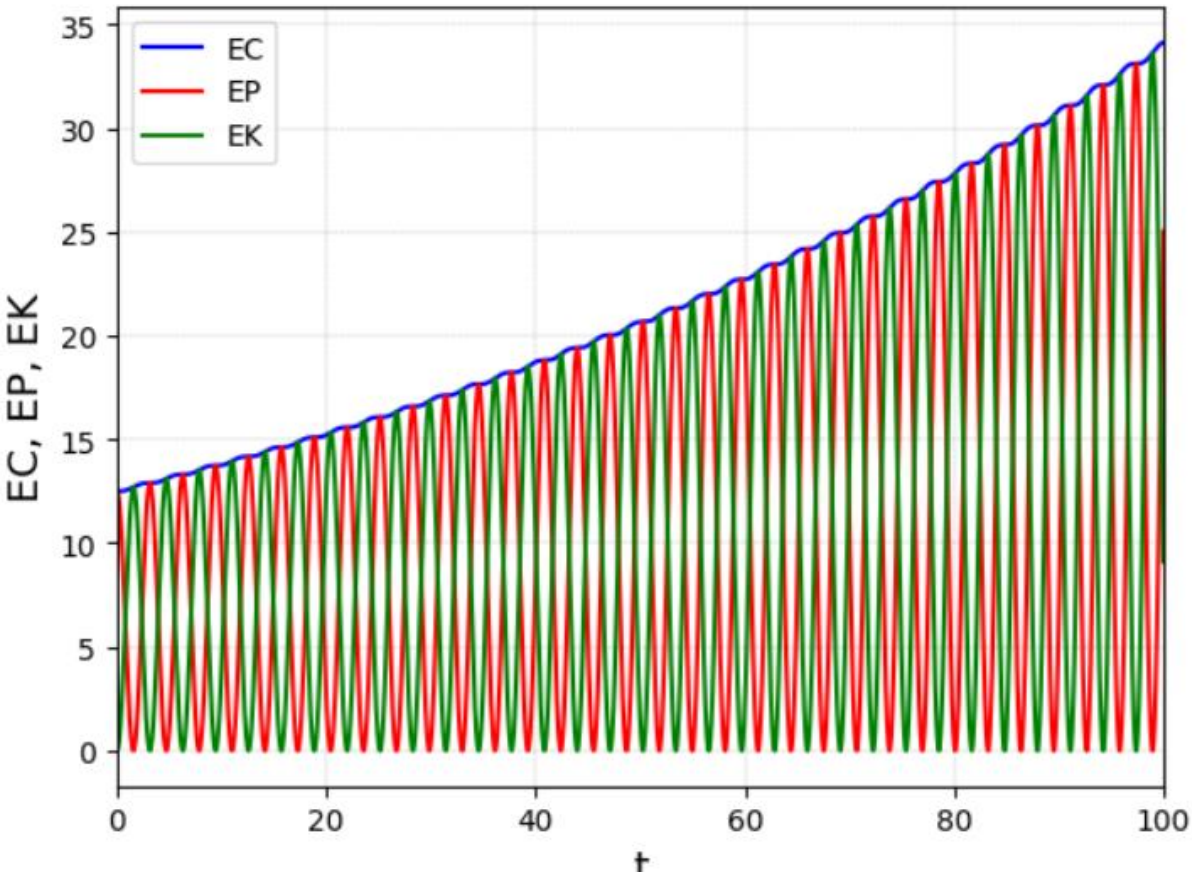





```
#predkosciowy

x_p = []
x_p.append(A)

v_p = []
v_p.append(0)


a_p = []
a_p.append(-omega*omega*x_l[0])

for i in range(1,len(t)):
  a_p.append(-omega*omega*x_p[i-1])
  x_p.append(x_p[i-1] + (dt * v_p[i-1]) + (((dt**2) * a_p[i-1] )/2))
  v_p.append(v_p[i-1] + (( dt * ( a_p[i-1] + a_p[i] ) ) / 2))

plt.plot(t, x_p, '-b', t, v_p, '-r', t, a_p, '-g')
plt.xlim(0, 10)
plt.ylim(-7, 7)
plt.xlabel('t', fontsize = '14')
plt.ylabel('x(t), v(t), a(t)', fontsize = '14')
```

```
plt.legend(['x(t)', 'v(t)', 'a(t)'])
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.2)
plt.show()

EK_p = []
EP_p = []
EC_p = []
for i in range(len(x_p)):
 EK_p.append(0.5*m*v_p[i]**2)
 EP_p.append(0.5*k*(x_p[i])**2)
 EC_p.append(EK_p[i]+EP_p[i])

plt.plot(t, EC_p, '-b', t, EP_p, '-r', t, EK_p, '-g')
plt.xlim(0, 100)
plt.xlabel('t', fontsize = '14')
plt.ylabel('EC, EP, EK', fontsize = '14')
plt.legend(['EC', 'EP', 'EK'])
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.2)
plt.show()
```





Zadanie 3
Rozwiązać numerycznie równanie różniczkowe tłumionego oscylatora harmonicznego korzystając ze
wszystkich poznanych algorytmów z rodziny Verleta (standard, leap-frog, prędkościowy). Wykonać wykresy
x(t), v(t), a(t), wykres przestrzeni fazowej p(x), sprawdzić zachowanie energii w czasie oraz odchylenia
standardowego energii całkowitej od kroku czasowego (w skali logarytmicznej). Uwzględnić fakt, że ze
względu na tłumienie, energia całkowita zmienia się

wzór na drgania tłumione

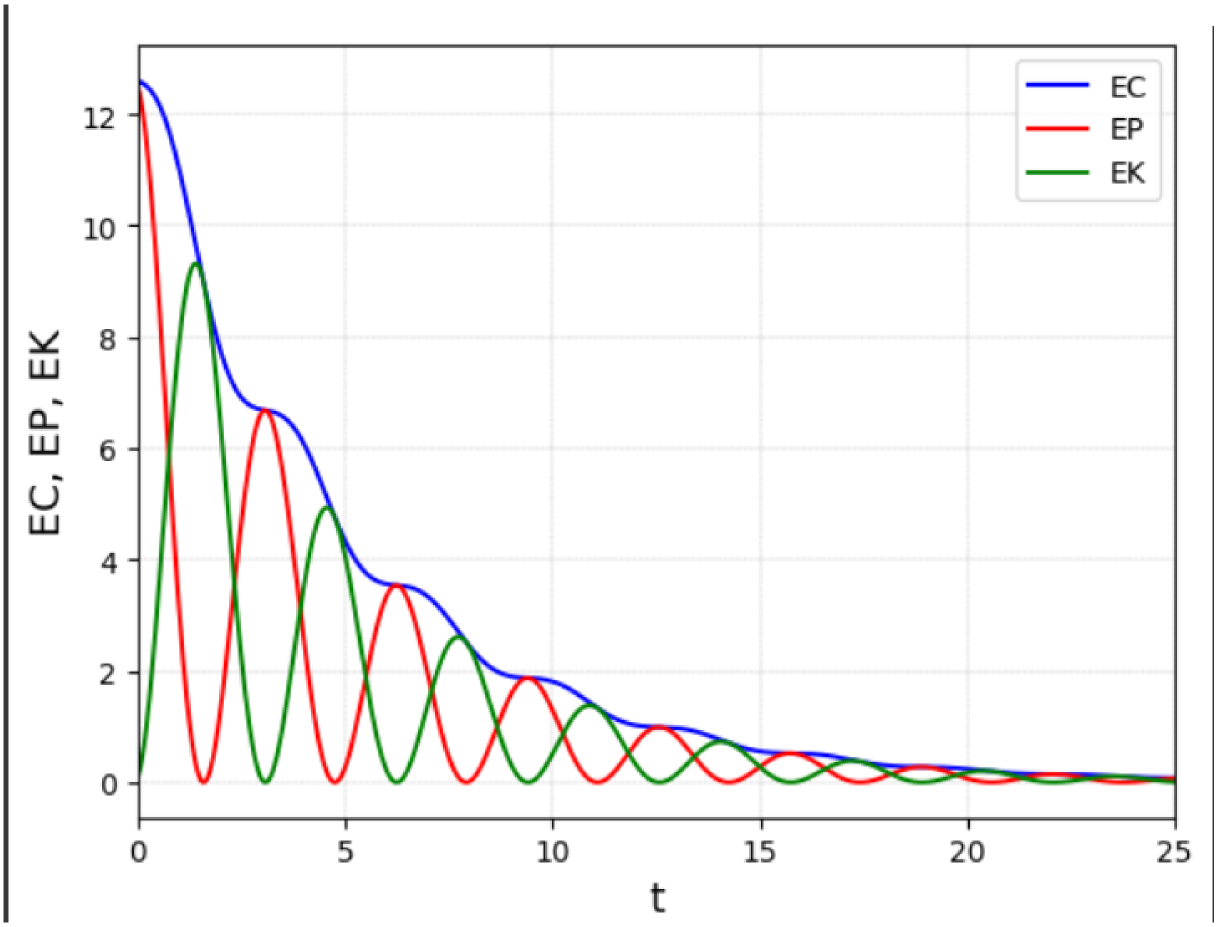
$$x(t) = Ae^{-\gamma t}\cos(\omega' t + \phi),$$

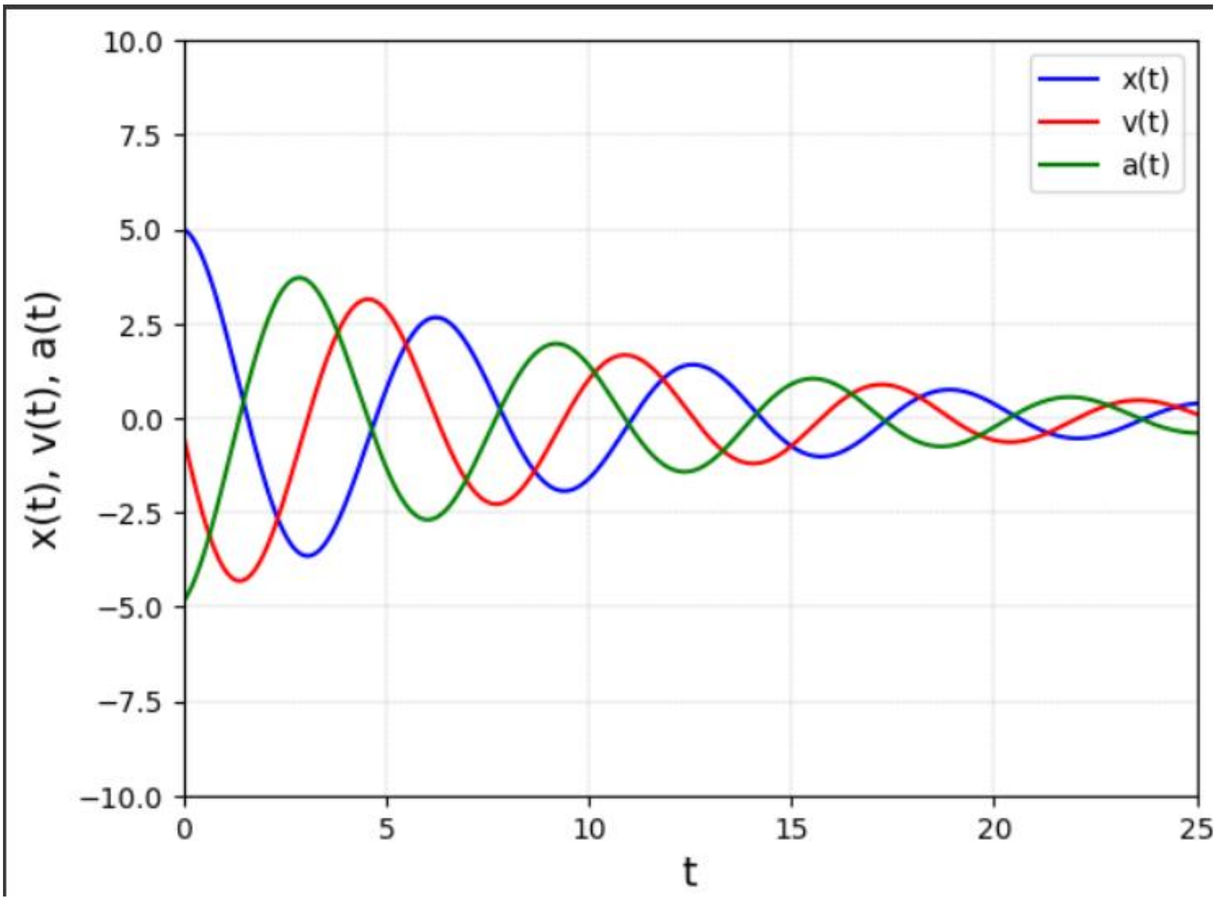
$$v(t) = -Ae^{-\gamma t}\gamma\cos(\omega' t + \phi) - Ae^{-\gamma t}\omega'\sin(\omega' t + \phi).$$

$$a(t) = Ae^{-\gamma t}(\gamma^2 - \omega'^2)\cos(\omega' t + \phi) - Ae^{-\gamma t}2\gamma\omega'\sin(\omega' t + \phi).$$

```
import math
import numpy as np
import matplotlib.pyplot as plt
A = 5
k = 1
m = 1
f = 1
omega = math.sqrt(k/m)
gamma = 0.1
dt = 0.01
t = np.arange(0, 100, dt)
omega2 = math.sqrt((omega * omega) - (gamma * gamma))

#polozenie
x_s = []
x_s.append(A)
x_s.append(A * math.exp(-gamma * dt) * math.cos(omega2 * dt))

#predkosc
v_s = []
v_s.append(0)
v_s.append((x_s[1] - x_s[0]) / (dt))

#przyspieszenie
a_s = []
a_s.append(-omega2 * omega2 * x_s[0] - 2 * gamma * v_s[0])
a_s.append(-omega2 * omega2 * x_s[1] - 2 * gamma * v_s[1])


#standardowe
for i in range(2, len(t)):
 x_s.append( 2 * x_s[i-1] - x_s[i-2] + (dt**2) * a_s[i-1])
 a_s.append(-omega2 * omega2 * x_s[i] - 2 * gamma * v_s[i-1])
 v_s.append((x_s[i] - x_s[i-2]) / (2*dt))

EK_s = []
EP_s = []
EC_s = []

for i in range(len(x_s)):
   EK_s.append(0.5 * m * v_s[i]**2)  # Energia kinetyczna
   EP_s.append(0.5 * k * x_s[i]**2)  # Energia potencjalna
   EC_s.append(EK_s[i] + EP_s[i])    # Energia całkowita
```

```
plt.plot(t, x_s, '-b', t, v_s, '-r', t, a_s, '-g')
plt.xlim(0, 25)
plt.ylim(-10, 10)
plt.xlabel('t', fontsize = '14')
plt.ylabel('x(t), v(t), a(t)', fontsize = '14')
plt.legend(['x(t)', 'v(t)', 'a(t)'])
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.2)
plt.show()

plt.plot(t, EC_s, '-b', t, EP_s, '-r', t, EK_s, '-g')
plt.xlim(0, 25)
#plt.ylim(-10, 10)
plt.xlabel('t', fontsize = '14')
plt.ylabel('EC, EP, EK', fontsize = '14')
plt.legend(['EC', 'EP', 'EK'])
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.2)
plt.show()
```

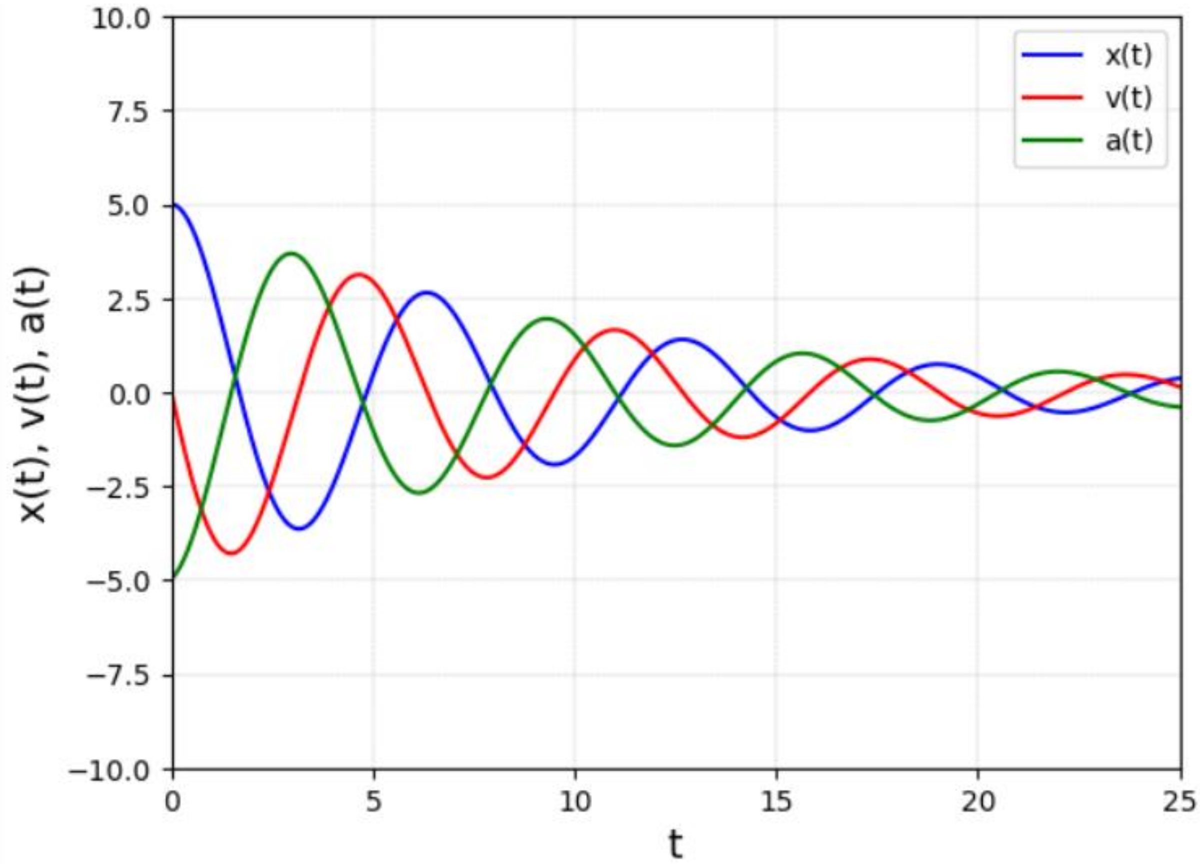


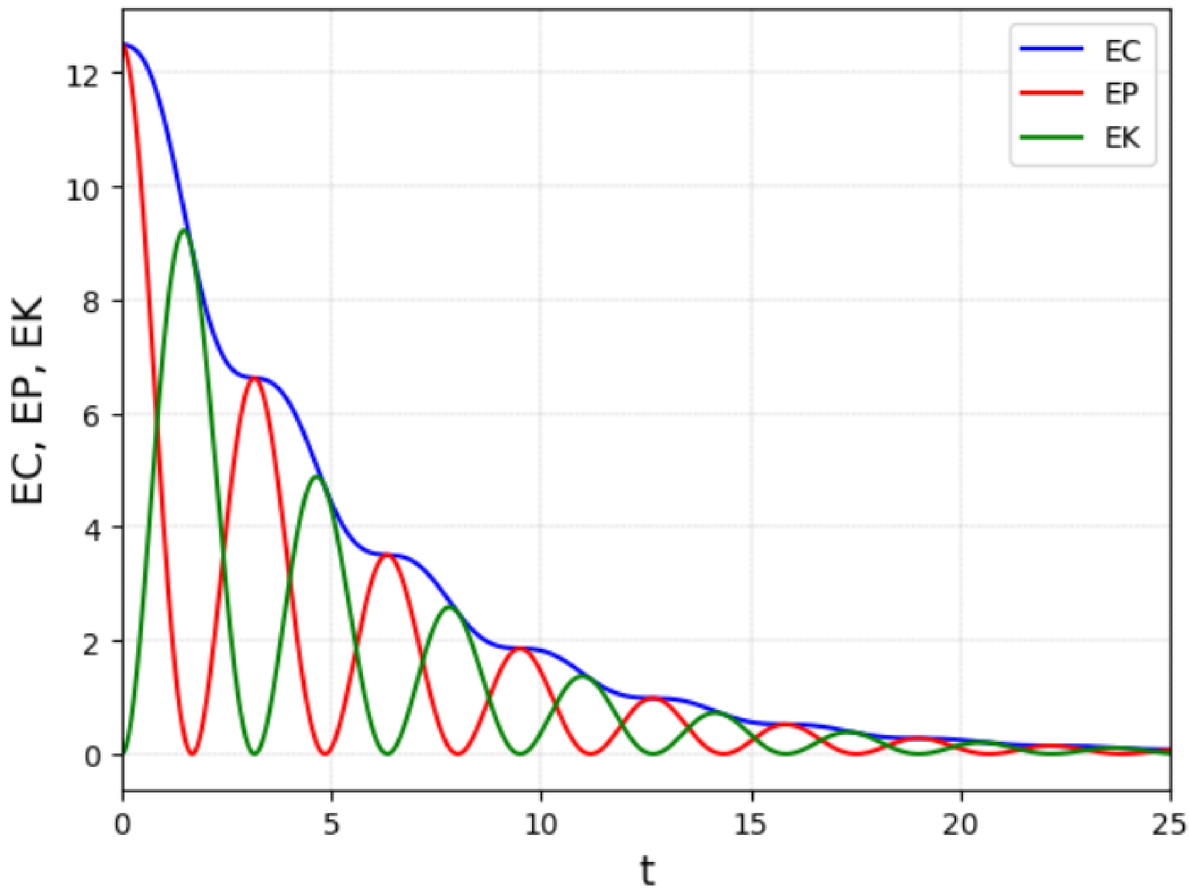


LEAP FROG

```
import math
import numpy as np
import matplotlib.pyplot as plt
A = 5
k = 1
m = 1
f = 1
omega = math.sqrt(k/m)
gamma = 0.1
dt = 0.01
t = np.arange(0, 100, dt)
omega2 = math.sqrt((omega * omega) - (gamma * gamma))

#polozenie
x_l = []
x_l.append(A)
#x_l.append(A * math.exp(-gamma * dt) * math.cos(omega2 * dt))

#predkosc
v_l = []
v_l.append(0)
```

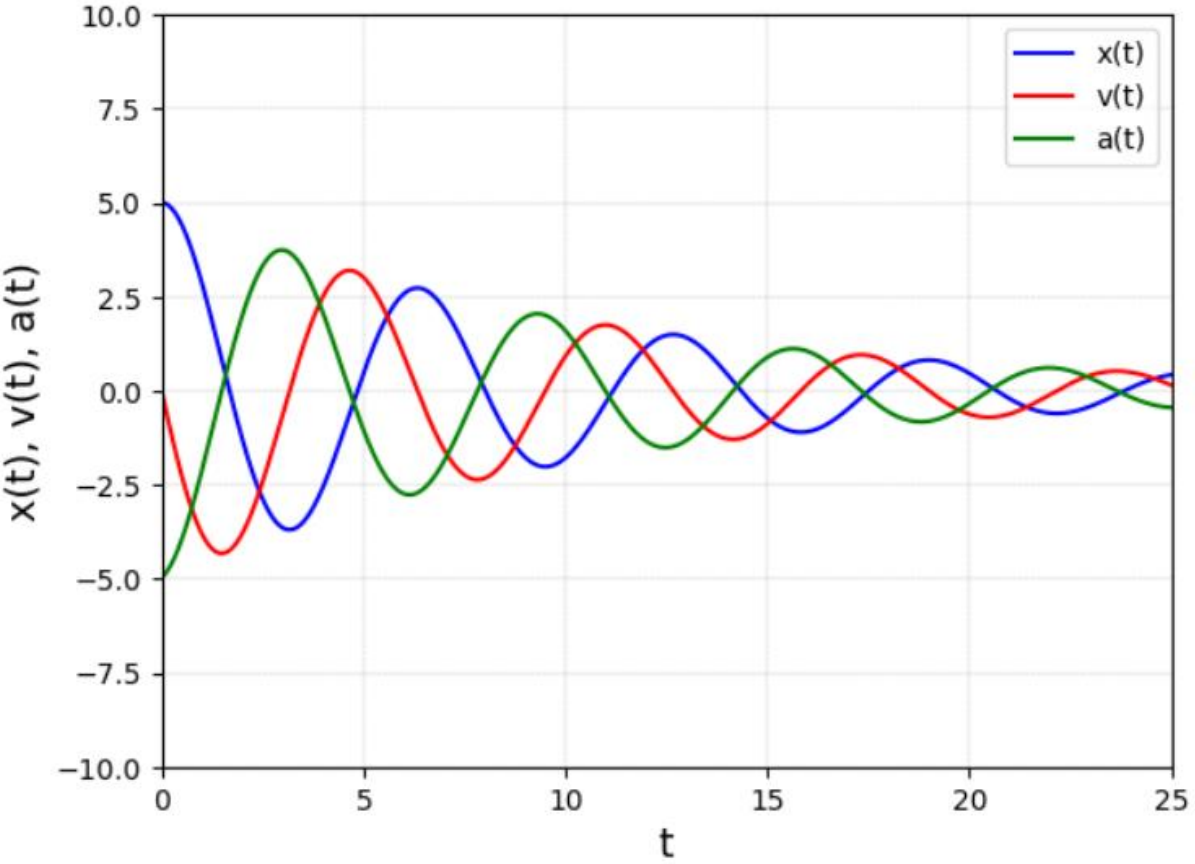
```
#v_l.append((x_s[1] - x_s[0]) / (dt))

#przyspieszenie
a_l = []
a_l.append(-omega2 * omega2 * x_l[0] - 2 * gamma * v_l[0])
#a_l.append(-omega2 * omega2 * x_l[1] - 2 * gamma * v_l[1])

#leap frog
for i in range(1, len(t)):
 a_l.append(-omega2 * omega2 * x_l[i-1] - 2 * gamma * v_l[i-2])
 v_l.append(v_l[i-1] + dt * a_l[i])
 x_l.append(x_l[i-1] + dt * v_l[i])


EK_l = []
EP_l = []
EC_l = []

for i in range(len(x_l)):
   EK_l.append(0.5 * m * v_l[i]**2)  # Energia kinetyczna
   EP_l.append(0.5 * k * x_l[i]**2)  # Energia potencjalna
   EC_l.append(EK_l[i] + EP_l[i])    # Energia całkowita



plt.plot(t, x_l, '-b', t, v_l, '-r', t, a_l, '-g')
plt.xlim(0, 25)
plt.ylim(-10, 10)
plt.xlabel('t', fontsize = '14')
plt.ylabel('x(t), v(t), a(t)', fontsize = '14')
plt.legend(['x(t)', 'v(t)', 'a(t)'])
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.2)
plt.show()

plt.plot(t, EC_l, '-b', t, EP_l, '-r', t, EK_l, '-g')
plt.xlim(0, 25)
#plt.ylim(-10, 10)
plt.xlabel('t', fontsize = '14')
plt.ylabel('EC, EP, EK', fontsize = '14')
plt.legend(['EC', 'EP', 'EK'])
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.2)
plt.show()
```


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
5
10
15
20
25
t
x(t), v(t), a(t)
x(t)
v(t)
a(t)

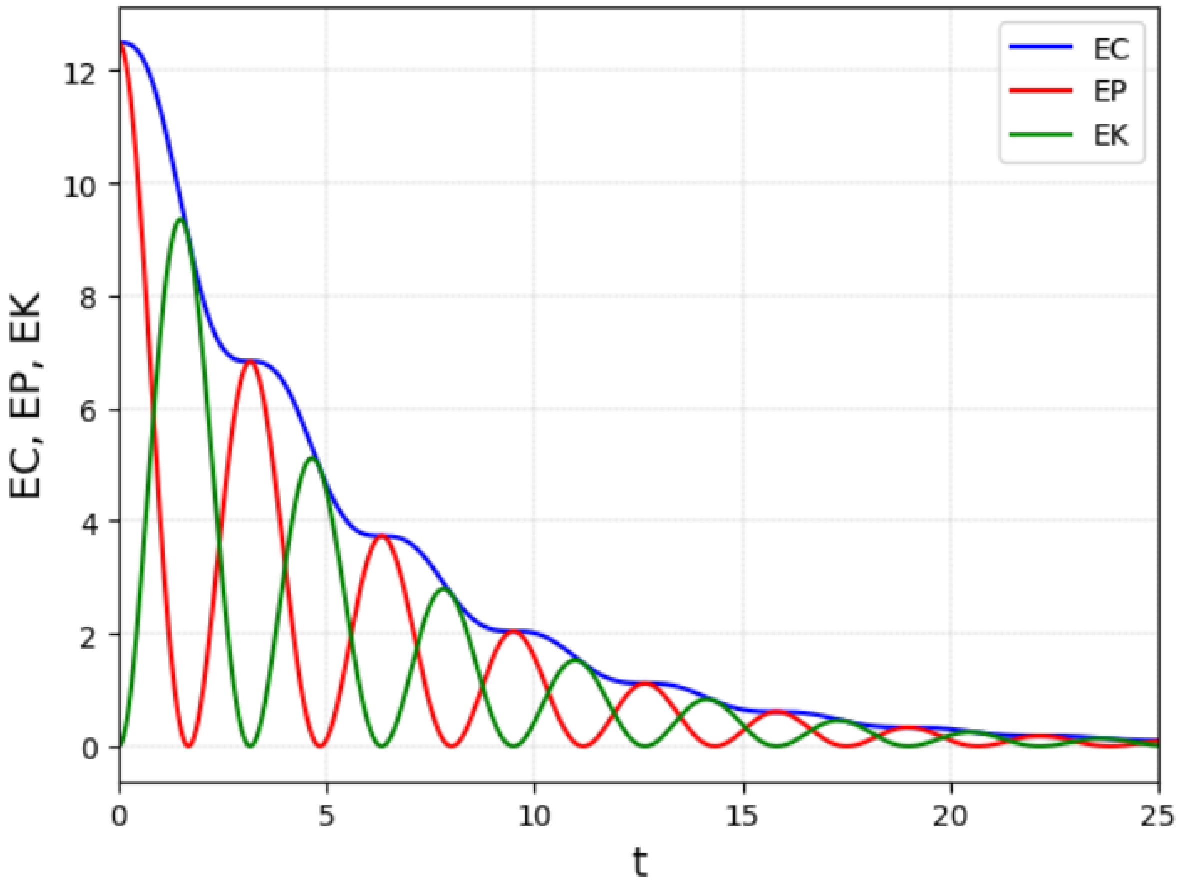

12
10
8
6
4
2
0
0
5
10
15
20
25
t
EC, EP, EK
EC
EP
EK

```
import math
import numpy as np
import matplotlib.pyplot as plt
A = 5
k = 1
m = 1
f = 1
omega = math.sqrt(k/m)
gamma = 0.1
dt = 0.01
t = np.arange(0, 100, dt)
omega2 = math.sqrt((omega * omega) - (gamma * gamma))

#polozenie
x_p = []
x_p.append(A)

#predkosc
v_p = []
v_p.append(0)

#przyspieszenie
a_p = []
a_p.append(-omega2 * omega2 * x_p[0] - 2 * gamma * v_p[0])

#leap frog
for i in range(1, len(t)):
 x_p.append(x_p[i-1] + dt * v_p[i-1] + (dt * dt * a_p[i-1] * 0.5) )
 a_p.append(-omega2 * omega2 * x_p[i-1] - 2 * gamma * v_p[i-2])
 v_p.append(v_p[i-1] + 0.5 * dt * (a_p[i-1] + a_p[i]))


EK_p = []
EP_p = []
EC_p = []

for i in range(len(x_p)):
   EK_p.append(0.5 * m * v_p[i]**2)  # Energia kinetyczna
   EP_p.append(0.5 * k * x_p[i]**2)  # Energia potencjalna
   EC_p.append(EK_p[i] + EP_p[i])   # Energia całkowita



plt.plot(t, x_p, '-b', t, v_p, '-r', t, a_p, '-g')
plt.xlim(0, 25)
plt.ylim(-10, 10)
plt.xlabel('t', fontsize = '14')
```

```
plt.ylabel('x(t), v(t), a(t)', fontsize = '14')
plt.legend(['x(t)', 'v(t)', 'a(t)'])
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.2)
plt.show()

plt.plot(t, EC_p, '-b', t, EP_p, '-r', t, EK_p, '-g')
plt.xlim(0, 25)
#plt.ylim(-10, 10)
plt.xlabel('t', fontsize = '14')
plt.ylabel('EC, EP, EK', fontsize = '14')
plt.legend(['EC', 'EP', 'EK'])
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.2)
plt.show()
```



x(t)
v(t)
a(t)
x(t), v(t), a(t)
t
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
5
10
15
20
25



EC
EP
EK
EC, EP, EK
t
12
10
8
6
4
2
0
0
5
10
15
20
25

WNIOSKI

Jak można zauważyć w przypadku zadania w zadaniu 2, algorytm standardowy daje podobny efekt co algorytm leap frog. Różnica jest stosunkowo mała na wykresach, prawie nie zauważalna. Natomiast w algorytmie prędkościowym widać jak wartości zaczynają iść ku górze.
Sytuacja ma się inaczej w przypadku zadania 3 gdzie, wyniki zaczynają się na wykresie pokrywać.
Warto jednak zwrócić uwagę że każdy algorytm służy do czego innego, gdzie na przykład do uzyskania prędkości najlepiej sprawdzi się a prędkościowy sposób porównując go do klasycznego gdzie możliwe jest tylko oszacowanie