Avtor: Gregor Podpečan, Vpisna številka: 23150289

Datum: 18.01.2020

Potrjujem, da sem avtor projektne naloge in da sem vso vsebino pripravil sam. V primeru, da se ugotovi plagiatorstvo se zavedam,da ne bom izpolnjeval pogojev za pristop k izpitu.

Kazalo

- 1 Inštalacija
- 2 Definicija naloge
- 3 Analiza meta z zračnim uporom.
  - 3.1 Definicija enačb
  - 3.2 Numerična analiza, resevanje diff. enacb
- 4 Simbolno reševanje
  - 4.1 Simbolni izračun sile upora
- 5 Interpolacija
- 6 Izračun ničle funkcije
- 7 Integracija
- 8 Sistem linearnih enacb
- 9 Aproximacija
- 10 Uporabniški vmesnik

Definicija naloge

Za projekt pri numeričnih metodah sem si izbral gibanje izstreljene žogice z upoštevanjem upora zraka. Ideja se mi je zdela primerna saj smo z silami, ki delujejo v tem primeru v stiku vsak dan. Priložena slika ponazarja pot izstrelka.

Vir:https://www.quora.com/While-considering-the-projectile-motion-the-effect-of-air-resistance-is-usually-ignored-If-however-air-resistance-is-not-ignored-what-is-its-effect-on-the-path-of-the-projectile (https://www.quora.com/While-considering-the-projectile-motion-the-effect-of-air-resistance-is-usually-ignored-If-however-air-resistance-is-not-ignored-what-is-its-effect-on-the-path-of-the-projectile) (18.01.2020)

Namen individualnega seminarja je prikazati ustrezno izbiro in uporabo numeričnih metod, komentirati njihovo delovanje in ovrednotiti dobljene rezultate. Reševanje našega fizikalnega problema mora zajemati sledeča poglavja:

- Simbolno reševanje
- Sistemi linearnih enačb
- Interpolacija ali aproksimacija
- Iskanje ničel
- Integriranje ali odvajanje
- Reševanje diferencialnih enačb

Analiza meta z zračnim uporom.

Definicija osnovnih enačb za analizo.

In [1]:

```
%matplotlib inline
%matplotlib notebook

from math import pi, cos, sin, tan
import matplotlib
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
```

Definicija enačb

$$F_{upora} = ma_{upora} = \frac{1}{2}c_v\rho v^2 S = \frac{1}{2}c_v\rho_z v^2 \pi r^2$$
$$F_{teze} = mg$$

$$F_y = F_{teze} + F_{upora_y}$$

$$F_x = F_{upora_x}$$

Zapisemo diferencijalne enacbe iz zgoraj definiranih katere bodo osnova za resevanje v nadaljevanju.

$$m\frac{d^2y}{dt^2} = -mg - \frac{1}{2}c_v\rho_z\left(\frac{dy}{dt}\right)^2\pi r^2$$
$$m\frac{d^2x}{dt^2} = -\frac{1}{2}c_v\rho_z\left(\frac{dx}{dt}\right)^2\pi r^2$$

Pri cemer naslednji spisek pojasnjuje obstoj simbolov v zgornjih enacbah...

- m = skupna masa zoge $(0.4kg)$
- g = gravitacijski pospesek $(9.81m/s^2)$
- ρ_z = gostota zraka $(1.2kg/m^3)$
- r = radij zoge $(0.1m)$
- c_v = sferni koelificijent upora (0.47)

Numerična analiza, resevanje diff. enacb

Za reševanje diferencialnih enačb bi lahko uporabili solve_ivp vendar je funkcija dokaj enostavna (brez visokih diferencialov). Zato rešimo diferencialne enačbe z iteracijo.

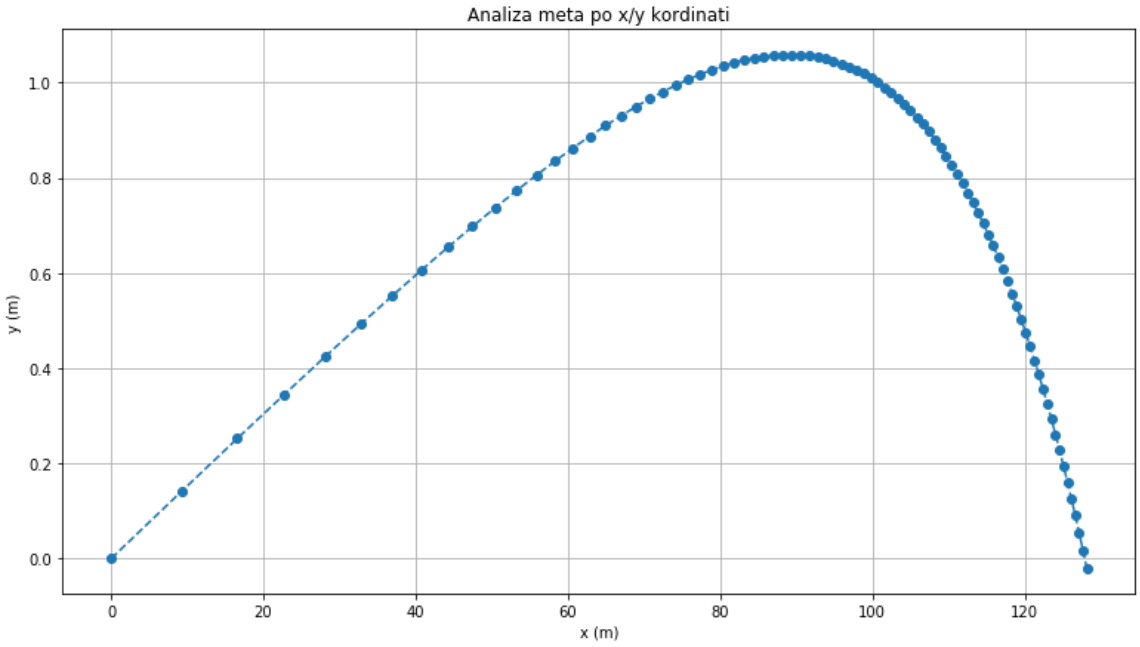
In [2]:

```
m = 0.4
g = 9.81
r = 0.1
Cv = 0.47
rho = 1.2
pi = 3.146

def upor(v): #Zaviralni pospesek vetra
    return (- 1/2 * Cv * rho * v**2 * pi * r**2)/m

# Definicija zacetnih pogojev
vx0 = 1300
vy0 = 20
x0 = 0
y0 = 0

# Definicija koraka
dt = 0.01

def simulacija_meta(dt, vx0, vy0, x0, y0):
    '''
    Izračun diferencialne enačbe leta izstrelka z upoštevanjem zračnega upora.
    Funkcija vrne vrednosti x in y koordinat izstrelka med letom vektorja hitrosti v sm
eri x in y ter časovne vrednosti
    pri koraku dt. Reševanje diferencialne enačbe zaključimo, ko je višina izstrelka ne
gativna.

    Argumenti:

    dt:  korak časa, ki ga iteracija uporabi za nov izračun
    vx0: začetna hitrost v x smeri
    vy0: začetna hitrost v y smeri
    x0:  začetna vrednost x (začnemo v izhodišču koordinatnega sistema)
    y0:  začetna vrednost y (začnemo v izhodišču koordinatnega sistema)


    '''

    # Lista izracunov
    vx = [vx0]
    vy = [vy0]
    x = [x0]
    y = [y0]
    t = [0]

    # Numericno resevanje
    while True:
        alfa = tan(vy[-1]/vx[-1]) # Naklon smeri hitrosti
        v = (vx[-1]**2 + vy[-1]**2)**0.5 # Vsota vektorja hitrosti
        u = upor(v) # Izracun upora vetra
        uX = u*cos(alfa) # Upor po x
        uY = u*sin(alfa) # Upor po y
        vx.append(vx[-1] + uX*dt) # Izracun nove hitrosti vetra po x
        vy.append(vy[-1] -g*dt + uY*dt) # Izracun nove hitrosti vetra po y
        x.append(x[-1] + vx[-1]*dt) # Izracun novih kordinat
        y.append(y[-1] + vy[-1]*dt) # Izracun novih kordinat
        t.append(t[-1]+dt)
        if y[-1]<0:
            return t, x, y, vx, vy
```

```
tN, x, y, vxN,vyN = simulacija_meta(dt, vx0, vy0, x0, y0)
```

In [3]:

```
plt.figure(figsize=(13,7))
plt.plot(x, y, linestyle='--', marker='o')
plt.xlabel('x (m)')
plt.ylabel('y (m)')
plt.title('Analiza meta po x/y kordinati')
plt.grid()
plt.show()
```



Simbolno reševanje

Termin simbolno računanje pomeni, da matematične izraze rešujemo strojno v obliki abstraktnih simbolov (in ne numerično). Strojno simbolno računanje nam pomaga kadar nas zanima rezultat v simbolni obliki in so izrazi preobsežni za klasično reševanje na list in papir. K strojnemu reševanju se zatečemo tudi zaradi zmanjšanja možnosti napake (pri obsežnih izračunih se ljudje lahko zmotimo).

Slavič, Janko.*Programiranje in numerične metode v ekosistemu Python* Ljubljana, 2020

In [4]:

```
sym.init_printing()

A1, A2, A3, t = sym.symbols("A_1, A_2, A_3, t")

m = 0.4        # Masa izstrelka [kg]
g = 9.81       # Gravitacijski pospešek [m/s^2]
rho = 1.2      # Gostota zraka [kg/m^3]
r = 0.1        # Polmer žogice (izstrelka) [m]
Cv = 0.47      # Koeficient upora krogle (izstrelka) [/]
PI = 3.146     # Vrednost pi
alpha = PI/4   # Kot pod katerim izstrelimo izstrelek
v_0 = 10       # Začetna hitrost

vx = v_0 * cos(alpha) # Hitrost v x smeri
vy = v_0 * sin(alpha) # Hitrost v y smeri

parametri_sym = {
    A1: m,
    A2: -m*g,
    A3: - (1/2) * Cv * rho * PI * r**2
}

display(parametri_sym)
```

$$\{A_1 : 0.4, \quad A_2 : -3.9240000000000004, \quad A_3 : -0.00887172\}$$

In [5]:

```
x_f = sym.Function('x')

x_diff_enacba = sym.Eq(A1*x_f(t).diff(t, 2), A3 * x_f(t).diff(t, 1)**2)

x_diff_enacba
```

Out[5]:

$$A_1 \frac{d^2}{dt^2} x(t) = A_3 \left(\frac{d}{dt} x(t) \right)^2$$

In [6]:

```
y_f = sym.Function('y')

y_diff_enacba = sym.Eq(A1*y_f(t).diff(t, 2), A2 + A3 * y_f(t).diff(t, 1)**2)

y_diff_enacba
```

Out[6]:

$$A_1 \frac{d^2}{dt^2} y(t) = A_2 + A_3 \left(\frac{d}{dt} y(t) \right)^2$$

In [7]:

```
x_diff_resitev = sym.dsolve(x_diff_enacba, x_f(t))
display(x_diff_resitev)
```

$$\left[x(t) = \frac{A_1}{A_3}\log\left(\frac{A_1}{A_3\left(C_1 + C_2 t\right)}\right), \quad x(t) = \begin{cases} C_1 t + C_2 & \text{for } A_3 = 0 \\ \text{NaN} & \text{otherwise} \end{cases}\right]$$

Simbolično bi diferencialno enačbo po y-u rešili z napisanim spodaj. Komentar: Ko sem projektno delal doma na PC-ju je delalo normalno, ko pa sem jo pisal v Ljubljani na laptopu pa kode ni izvršilo zato sem jo dodal ko komentar.

In [8]:

```
#y_diff_resitev = sym.dsolve(y_diff_enacba, y_f(t))
#display(y_diff_resitev)
```

Simbolni izračun sile upora

Zanima nas sila upora, ki deluje na žogo med letom oziroma pri začetni hitrosti, saj vemo, da se hitrost žoge med letom zmanjšuje.

- m = skupna masa zoge $(0.4kg)$
- ρ_z = gostota zraka $\left(1.2kg/m^3\right)$
- r = radij zoge $(0.1m)$
- c_v = sferni koeficient upora (0.47)
- v_0 = začetna hitrost zoge (1300) #Še enote

Enačbaza presek žoge

$$S = \pi r^2$$

Enačba za silo upora žoge

$$F_{upora} = ma_{upora} = \frac{1}{2}c_v\rho v^2 S = \frac{1}{2}c_v\rho_z v^2 \pi r^2$$

In [9]:

```
# Z uporabo simbolnega računanja pripravimo enačbo za silo upora.
from sympy.abc import rho
F, Cv, rho, v_0 , pi, r = sym.symbols("F, Cv, rho, v_0 , pi, r")
sila_upora = sym.Eq(F,1/2 * Cv * rho * v_0**2 * sym.pi * r**2)
sila_upora
```

Out[9]:

$$F = 0.5\pi C v r^2 \rho v_0^2$$

In [10]:

```
#Izračunamo silo upora.
parametri = {Cv : 0.47, rho : 1.2, v_0 : 10, r : 0.1}
sila_upora_res = sila_upora.subs(parametri).evalf(n=5)
display(sila_upora_res)
```

$F = 0.88593$

Interpolacija

Pri interpolaciji izhajamo iz tabele različnih vrednosti, določiti pa želimo vmesne vrednosti. V okviru interpolacije točke povežemo tako, da predpostavimo neko funkcijo in dodamo pogoj, da mora funkcija potekati skozi točke.

Slavič, Janko.*Programiranje in numerične metode v ekosistemu Python* Ljubljana, 2020

In [11]:

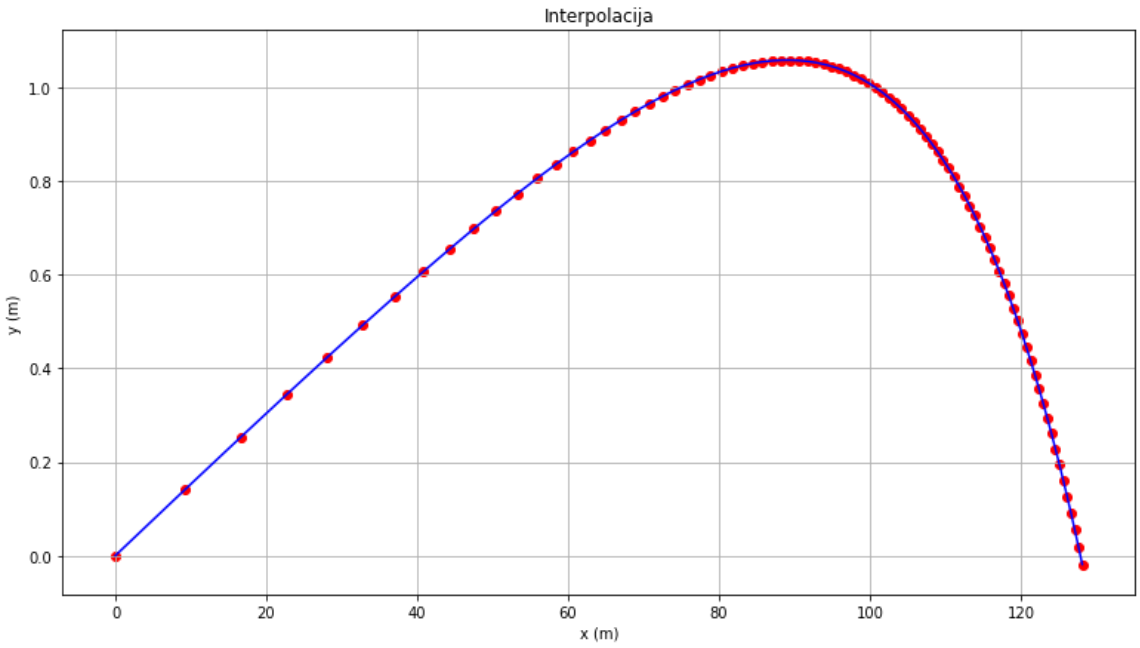
```
from scipy.interpolate import interp1d
```

In [12]:

```
interpolacija = interp1d(x, y, kind = "quadratic", fill_value="extrapolate")
```

In [13]:

```
plt.figure(figsize=(13,7))
plt.plot(x, y, color='b')
plt.scatter(x, interpolacija(x), color = "r", marker = "o", )
plt.xlabel("x (m)")
plt.ylabel("y (m)")
plt.title("Interpolacija")
plt.grid()
plt.show()
```



Izračun ničle funkcije

V okviru reševanja enačb obravnavamo poljubno enačbo, ki je odvisna od spremenljivke x in iščemo rešitev: f(x) = 0. Rešitvam enačbe rečemo tudi koreni. Koren enačbe f(x) = 0 je hkrati tudi ničla funkcije y = f(x).Funkcija y = f(x) ima lahko ničle stopnje:

- Ničla prve stopnje: Funkcija seka abcisno os pod neničelnim kotom,
- Ničla sode stopnje: funkcija se dotika abcisne osi, vendar je ne seka,
- Ničla lihe stopnje: funkcija seka abcisno os, pri ničli stopnje 3 in imamo prevoj(tangenta je vzporedna z abcisno osjo).

Slavič, Janko.*Programiranje in numerične metode v ekosistemu Python* Ljubljana, 2020

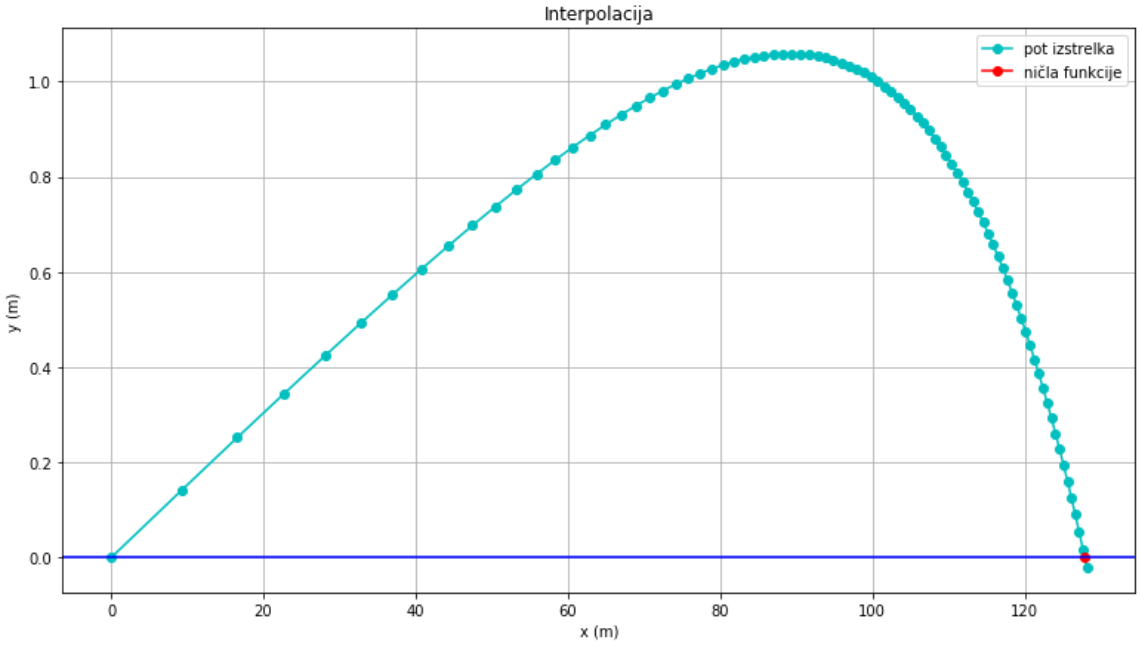
In [14]:

```
from scipy.optimize import bisect

nicla = bisect(interpolacija, 120, 140)

print("----------------------")
print(f"Nicla najdena: {nicla}")
```

```
----------------------
Nicla najdena: 127.8950387122461
```

In [15]:

```
plt.figure(figsize =(13,7))
plt.plot(x, interpolacija(x), color = "c", marker = "o", label = "pot izstrelka")
plt.plot(nicla, 0, color = "r", marker = "o", label = "ničla funkcije")
plt.xlabel("x (m)")
plt.ylabel("y (m)")
plt.title("Interpolacija")
plt.legend()
plt.grid()
plt.axhline(0, color = "b")
plt.show()
```



Integracija

V okviru tega poglavja bomo za dano funkcijo v_x, v_y izračunali pospešek preko gradienta teh funkcij:

$$a_x = \frac{d}{dt} v_x(t)$$

$$a_y = \frac{d}{dt} v_y(t)$$

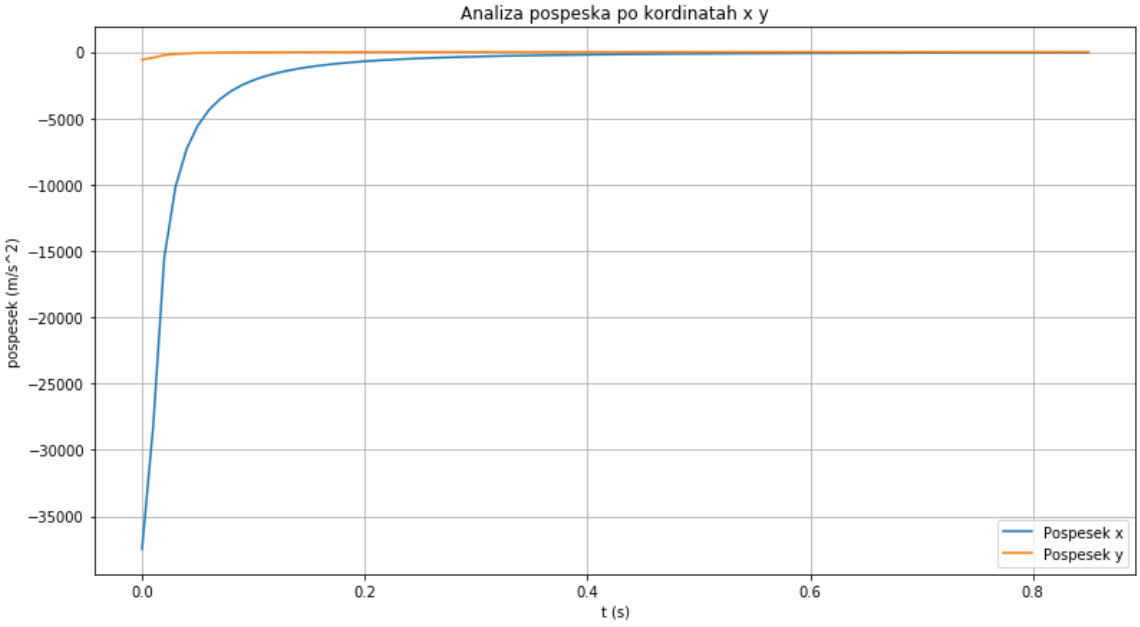
$$a = \sqrt{(a_x^2 + a_y^2)}$$

kjer pa so vrednosti funkcije, ki jih pridobimo iz tabele vrednosti ali s pomočjo analitične funkcije.

Slavič, Janko.*Programiranje in numerične metode v ekosistemu Python* Ljubljana, 2020

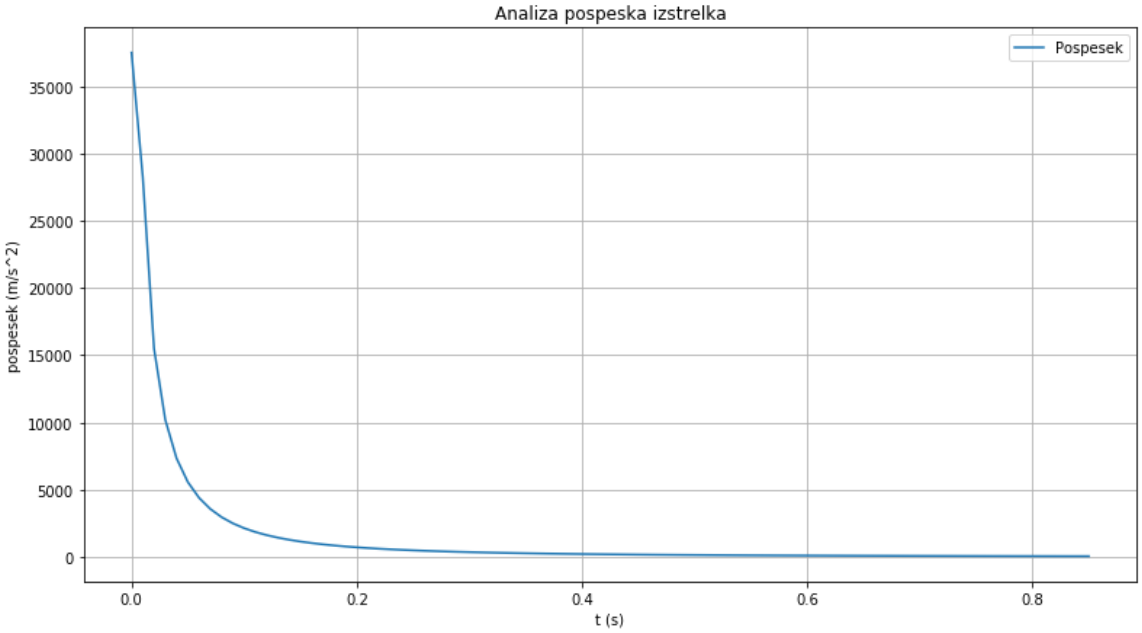
In [16]:

```
x_pospesek = np.gradient(vxN, tN)
y_pospesek = np.gradient(vyN, tN)

plt.figure(figsize=(13,7))
plt.plot(tN, x_pospesek, label="Pospesek x")
plt.plot(tN, y_pospesek, label="Pospesek y")
plt.title('Analiza pospeska po kordinatah x y')
plt.ylabel('pospesek (m/s^2)')
plt.xlabel('t (s)')
plt.grid()
plt.legend()
plt.show()
```



In [17]:

```
pospesek = (x_pospesek**2 + y_pospesek**2)**(1/2)

plt.figure(figsize=(13,7))
plt.plot(tN, pospesek, label="Pospesek")
plt.title('Analiza pospeska izstrelka')
plt.ylabel('pospesek (m/s^2)')
plt.xlabel('t (s)')
plt.grid()
plt.legend()
plt.show()
```



Sistem linearnih enacb

Editor latex (https://www.codecogs.com/latex/eqneditor.php)

Sistem enačb lahko zapišemo tudi v matrični obliki:

$$Ax = b$$

kjer sta A in b znana matrika in vektor, vektor x pa ni znan. Matriko A imenujemo matrika koeficientov, vektor b imenujemo vektor konstant in vektor x vektor neznank.

Slavič, Janko.*Programiranje in numerične metode v ekosistemu Python* Ljubljana, 2020

Z uporabo sistema linearnih enačb bom iskal hitrost v najvišji točki meta, nato pa jo bom primerjal z izračunano vrednostjo, ki sem jo pridobil z iterativnim izračunom diferencialnih enačb.

Najprej potrebujemo najvišjo točko meta.

In [18]:

```
max_y = max(y)
max_y
```

Out[18]:

$$1.0583286657057684$$

In [19]:

```
# Poiščemo indeks najvišje točke.
indeks = y.index(max_y)
indeks
```

Out[19]:

$$32$$

In [20]:

```
# Poiščemo še x koordinato najvišje točke.
x_hmax = x[32]
x_hmax
```

Out[20]:

$$89.40573043728297$$

In [21]:

```
kot = (np.tan(vxN[32]/vyN[32]))*np.pi/180
kot
```

Out[21]:

$$-0.03553789984005052$$

In [ ]:

```

```

In [22]:

```
# Izračunamo koeficiente linearne funkcije.
koef_123 = np.polyfit(tN, x, deg=1)
koef_123_1 = koef_123[0]
koef_123_2 = koef_123[1]
print(koef_123)
```

```
[117.49891428  41.72206143]
```

In [23]:

```
# Simbolično prikažemo funkcijo.
x_1, t = sym.symbols("x_1, t")
func = sym.Eq(x_1, koef_123_1*t + koef_123_2)
func
```

Out[23]:

$$x_1 = 117.498914284591t + 41.7220614322062$$

In [24]:

```
# Imamo koeficiente in ustvarimo objekt polinom.
polinom = np.poly1d(koef_123)
polinom
```

Out[24]:

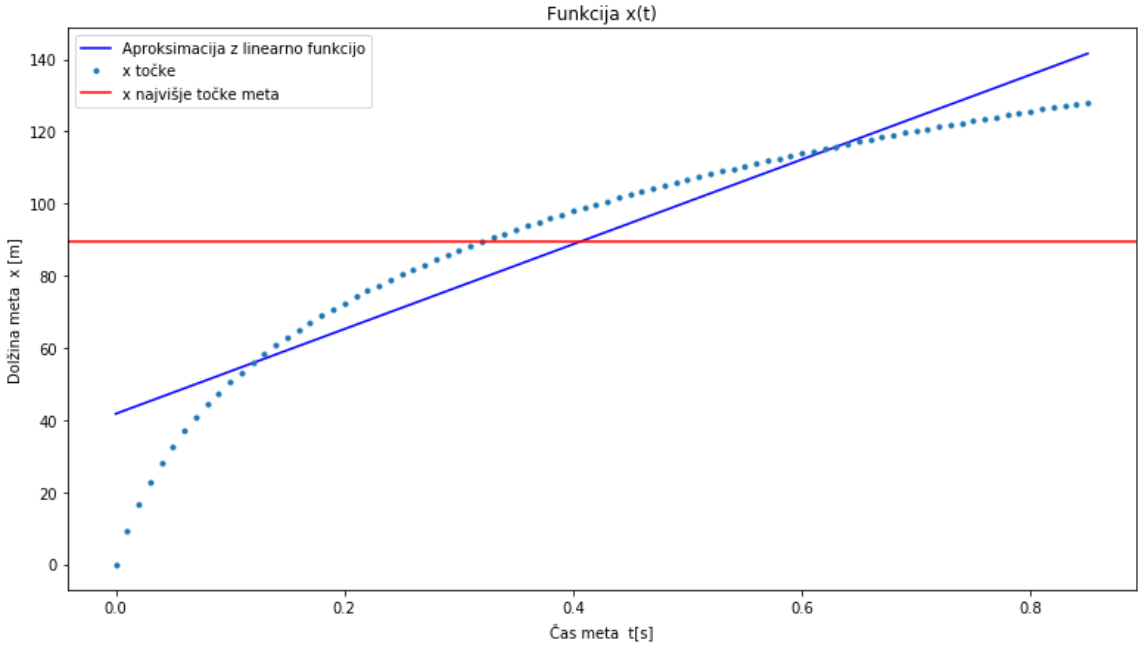
```
poly1d([117.49891428,  41.72206143])
```

In [25]:

```
# Graf lege x v odvisnosti od časa t
plt.figure(figsize=(13,7))
plt.plot(tN, polinom(tN), "b", label = "Aproksimacija z linearno funkcijo")
plt.plot(tN, x, ".", label="x točke")
plt.axhline(x_hmax, color= "r", label = "x najvišje točke meta")
plt.title("Funkcija x(t)")
plt.xlabel("Čas meta  t[s]")
plt.ylabel("Dolžina meta  x [m]")
plt.legend()
```

Out[25]:

```
<matplotlib.legend.Legend at 0x1ab2ee6eef0>
```



Definirajmo nas sistem v matricni obliki da ga bomo lahko resili z linearnim resevalcem...

$$\begin{bmatrix} 41.72 \\ 89.41 \end{bmatrix} = \begin{bmatrix} 117.50 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix}$$

In [26]:

```
# Zapišemo sistem linearnih enačb

A = np.array([[koef_123_1, 1],
              [0, 1]])

b = np.array([koef_123_2, x_hmax])
print(A)
print(b)
```

```
[[117.49891428   1.        ]
 [  0.           1.        ]]
[41.72206143 89.40573044]
```

In [27]:

```
# Izračunamo presečišče.

presecisce = np.linalg.solve(A, b)
presecisce
```

Out[27]:

```
array([-0.40582221, 89.40573044])
```

In [28]:

```
# Definiramo x koordinate funkcije f(x).

x_meta = polinom(tN)
```

In [29]:

```
# Določimo korak časa

h_cas = tN[2] - tN[1]
h_cas
```

Out[29]:

0.01

In [30]:

```
# Hitrost x je odvod funkcije po času.

odvod_s_t = np.gradient(x_meta, h_cas)
odvod_s_t
```

Out[30]:

```
array([117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428, 117.49891428, 117.49891428,
       117.49891428, 117.49891428])
```

In [31]:

```
# Hitrost po koordinatah

VX = 117.50
VY = VX*np.tan(kot*np.pi/180)
VY
```

Out[31]:

$$-0.07287977931667264$$

In [32]:

```
# Izračunamo skupno hitrost

iskana_hitrost = np.sqrt(VX**2 + VY**2)
print(f"Skupno hitrost: {round(iskana_hitrost,2)} m/s")
```

```
Skupno hitrost: 117.5 m/s
```

In [33]:

```
# Izračunamo hitrost, ki smo jo izračunali z diferencialnimi enačbami

izračunana_hitrost = np.sqrt(vxN[32]**2 + vyN[32]**2)
izračunana_hitrost
```

Out[33]:

118.53210541316892

In [34]:

```
# Preverimo koliko se izračunani hitrosti razlikujeta

razlika_hitrosti = izračunana_hitrost - iskana_hitrost
razlika_hitrosti
```

Out[34]:

1.0320828112041482

In [35]:

```
procent_razlike = (100*razlika_hitrosti)/izračunana_hitrost
procent_razlike
```

Out[35]:

0.8707200531084837

Vidimo da se hitrosti razlikujejo za manj kot procent, zato se mi zdi rešitev smiselna glede na to, da smo jo aproksimirali z linearno funkcijo.

Aproximacija

Kot dodatek projektni nalogi sem izbral aproksimacijo z uporabo CSV (comma separated values). Podatke sem uvozil z uporabo CSV-ja. Podatke sem pridobil na internetu. Dodal sem jih v mapo ter jih malce skrajšal nato pa sem jih aproksimiral z linearno funkcijo, da lahko vidimo trend, ki ga vrednost zlata ubira.

In [36]:

```
import csv
aprox_x = []
aprox_y = []
with open('annual_csv.csv', newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=',')
    for row in spamreader:
        aprox_x.append(int(row[0].split('-')[0]))
        aprox_y.append(float(row[1]))
```

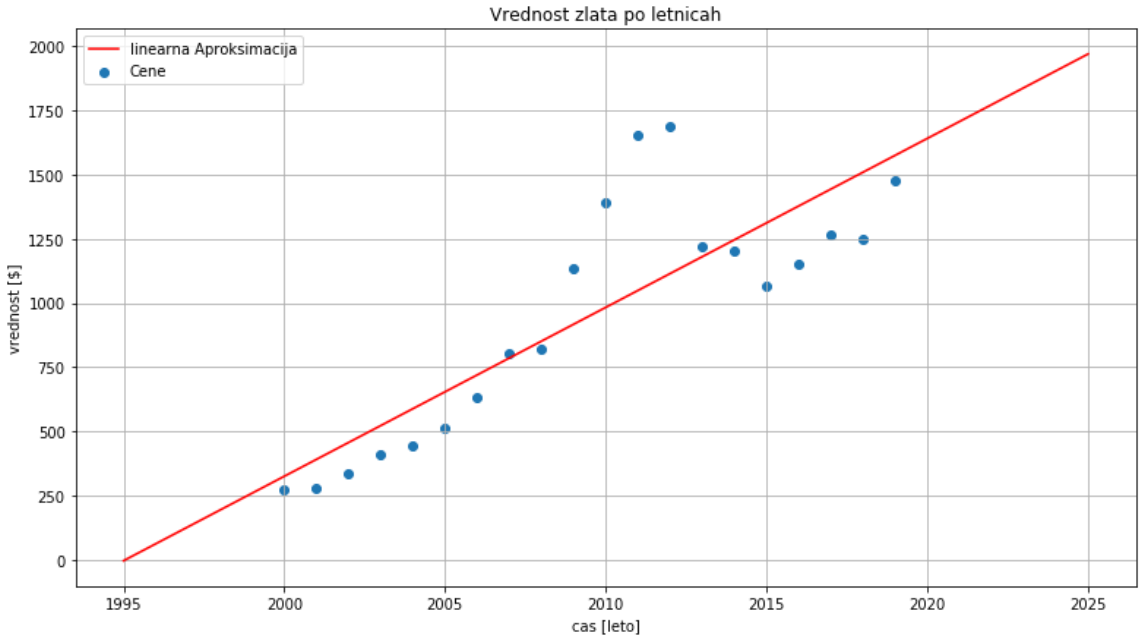
In [37]:

```
koelif = np.polyfit(aprox_x, aprox_y, deg=1)
polinom_fit = np.poly1d(koelif)
display(polinom_fit)
```

```
poly1d([ 6.58591436e+01, -1.31393898e+05])
```

In [38]:

```
letnice = np.linspace(1995, 2025, 500)
plt.figure(figsize=(13,7))
plt.plot(letnice, polinom_fit(letnice), label="linearna Aproksimacija",color='r')
plt.scatter(aprox_x, aprox_y, label="Cene")
plt.title("Vrednost zlata po letnicah")
plt.ylabel('vrednost [$]')
plt.xlabel('cas [leto]')
plt.grid()
plt.legend()
plt.show()
```



Uporabniški vmesnik

V primeru da analiziramo pojav za neko podjetje je zelo primerno da se naredi GUI (graphical user interface) s katerim lahko manj izskušeni inžinerji uporavljajo naše izračune in logiko analize za generične primere sorodnih tematik. Uporavljal bom uporabniški vmesnik ki ga ponuja knjižnica matplotlib.

V nasem primeru sem ustvaril GUI v katerem lahko uporabnik spreminja parametre posevnega meta z upostevanim primerom v katerem lahko spreminja zacetno v_x in v_y hitrost... Uporabniski vmesnik tudi prikaze maksimalen doseg po y in x smeri kakor tudi prikaze cas potovanja...

V istem direktoriju se nahaja modul `GUI.py` v katerem sem vnesel primer uporabniskega vmesnika. Da lahko pozenes uporabniski vmesnik je potrebno instalirati knjiznico `pyqtgraph` .

```
pip install pyqtgraph
```

Potem pa lahko pozenes naslednjo vrstico da se odpre uporabniski vmesnik...

In [ ]:

```
import GUI
GUI.main()
```