NUMERIČKO MODELIRANJE

PRIRUČNIK ZA UČENIKE

ERASMUS+ project "Green Science Teaching Materials for Digital Learning" (2022. - 2025.)

SADRŽAJ

Što je numeričko modeliranje i zašto je bitno?

Iako znamo da nema znanosti bez eksperimenta, nije uvijek moguće jednostavno provesti eksperiment da bismo potvrdili ili opovrgnuli postavljenu hipotezu. No i dalje prinicipi znanstvene discipline koji čine znanost onime što ona je zahtjevaju provođenje eksperimenta. Koliko god da je osmišljavanje i provođenje eksperimenta važan dio znanstvene dicipline, još važnije je objasniti njegove rezultate. Čak i onda kada je eksperiment moguće provesti najčešće kao rezultat dobivamo velike količine, na prvi pogled nepovezanih, podataka koje je potrebno analizirati i svemu dati nekakav smisao. Matematika nam daje puno načina na koje se eksperimentalni podaci mogu analizirati, ali to mogu biti dugotrajni procesi. S pojavom i razvojem računala to se promijenilio jer računala mogu brže i preciznije analizirati velike skupine podataka. I kada te sve probleme i njihova rješenja spojimo zajedno, dobivamo ono što je poznato kao tehnika numeričkog modeliranja. To je računalna tehnika koja se koristi za simulacije i analize kompleksnih sustava ili procesa korištenjem matematičkih modela i računalnih simulacija. Danas se ona svakodnevno koristi u priodnim i društvenim znanostima te inženjerstvu kako bi se predvidjelo ponašanje kompleksnih sustava te dobio uvid u njihovo ponašanje u različitim uvjetima. Numeričko modeliranje nam omogućava analizu onih sustava koje je teško ili nemoguće analizirati koristeći se tradicionalnim eksperimentom, ali i u slučajevima kada je to moguće ono je korisno za optimizaciju eksperimenta prije njegovog provođenja tradicionalnim putem (na taj način se smanjuju pogreške kod provođenja fizičkog eksperimenta te štedi vrijeme i resursi).

Zašto učiti tehniku numeričkog modeliranja?

Ova tehnika je sastavni dio moderne znanosti i svaki budući znanstvenik će se tijekom svoje karijere susresti s njom. Zašto onda ne početi što je ranije moguće i budućim znanstvenicima dati uvid u način na koji moderna znanost funkcionira. Numeričko modeliranje se temelji na interdisciplinarnosti - kod numeričkog modeliranja nekog sustava/problema istovremeno se koriste matematička i računalna znanja u svrhu opisa konkretnog problema unutar određene prirodne ili društvene znanosti. Na taj način primjenjujemo i povezujemo znanja iz područja koja inače gledamo zasebno što nam pomaže u razvoju divergentnog razmišljanja. Također, za primjenu principa numeričkog modeliranja potrebno je koristiti znanstveni pristup problemu koji uključuje postavljanje hipoteze, modeliranje situacije/sustava (zadržavanje onog bitnog za konkretan problem), traženje najboljeg načina za rješavanje problema, analizu rezultata i usporedbu ili primjenu na problemima slične vrste. Iako kod numeričkog modeliranja ne stavljamo naglasak na postavljanje hipoteze već na način ispitivanja hipoteza (jer se ona koristi kao alat), ova tehnika i dalje omogućava uvid u znanstvenu metodu i probleme s kojima se znanstvenici susreću tijekom tog procesa.

Što korisititi za numeričko modeliranje?

Postoji mnogo alata koji se mogu koristiti za numeričko modeliranje, od programa koje svakodnevno koristimo za obradu podataka poput MS Ecxel ili LibreCalc Office, do naprednijih računalnih softvera kao što su Wolfram Mathematica, MatLab ili slični softveri kao alternativa Wolfram Mathematici. Dobar dio tih naprednijih matematičkih softvera koje možemo korisitit i za numeričko modeliranje se plaća, ali postoje i besplatne alternative. Jedan od najčešće korištenih besplatnih matematičkih softvera je SageMath. SageMath je besplatan i otvoren matematički softver koji u sebi ima integrirane mnoge otvorene pakete poput NumPy, SciPy, matplotlib, SymPy i još mnogo drugih (gotovo 100 paketa). Sučelje za korisnika je bilježnica koja se otvara u web pregledniku ili komandnoj liniji. Bilježnica koju koristi SageMath je Jupyter Notebook koja podržava više od 40 programskih jezika, a jedan od njih je i Python koji je odličan za sve potrebe numeričkog modeliranja. Unutar bilježnice možete pisati tekst i kod, stvarati grafike, pisati jednadžbe i sve dijeliti s ostalim korisnicima. Više o SageMathu i Jupyter Notebooku možete pronaći na njihovim mrežnim stranicama www.sagemath.org (https://www.sagemath.org/index.html) i www.jupyter.org (https://jupyter.org/).

Chapter 1: What do we need to know before starting numerical modeling?

> *"Slika govori više od tisuću riječi."*

Bitan način vizualizacije nekog problema je upravo grafička reprezentacija podataka. Iako podatke uglavnom prikazujemo unutar tablica, grafovi su puno snažniji način reprezentacije podataka. Pomoću grafova se može prikazati velika količina podataka, a dobra grafička reprezentacija nekog problema nam pomaže kod jednostavnije interpretacije podataka. Korištenjem računala i računalnih programa je crtanje grafova postalo puno jednostavnije i brže, pogotovo kada se radi o velikom broju podataka. Neki problemi se vrlo jednostavno mogu riješiti ako ih dobro grafički prikažemo.

1.1. Crtanje grafova pomoću *SageMath*a

Crtanje grafova unutar SageMatha ćemo prikazati na nekoliko jednostavnih primjera.Pomoću SageMatha neku funkciju možemo grafički prikazati koristeći fukciju `plot()` unutar koje možemo koristiti i neke dodatne argumente.

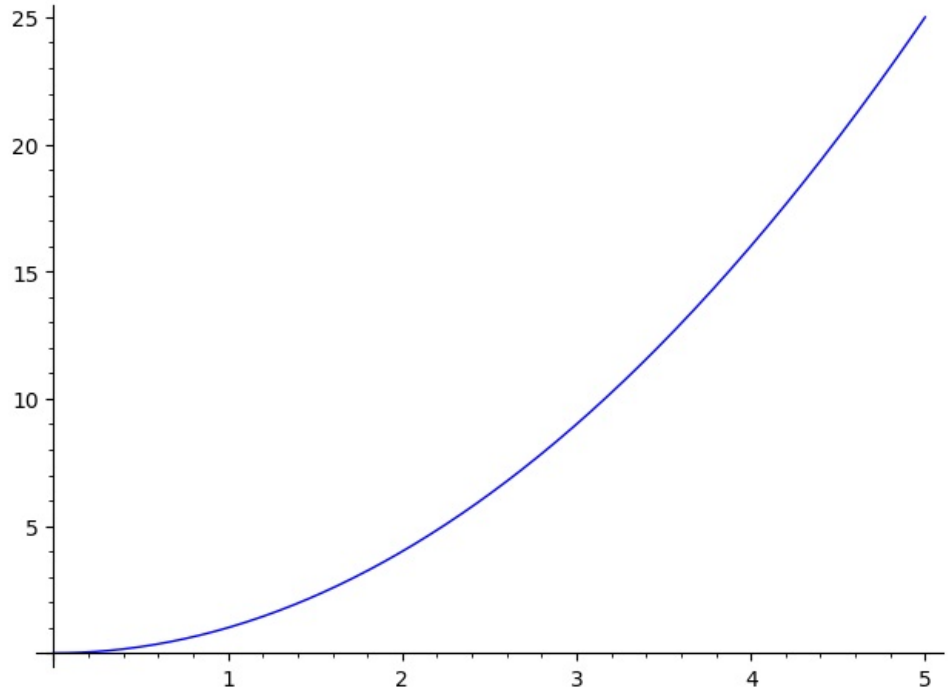
Primjer 1.1.

Nacrtajimo graf funkcije x^2 za *x* u intervalu $[0, 5]$.

In [1]:

```
plot(x^2,(x,0,5))
```

Out[1]:



Ako želimo nadograditi naš graf, potrebno je navesti i argumente. Neki od argumenata koji najčešće koristimo su navedeni u tablici.

title=''	**axes_labels=[]**	**color=''**	**thickness**	**linestyle**	**legend_label=''**
naslov grafa	nazivi osi\	boja krivulje\	debljina krivulje\	izgled linije\	naziv funkcije na legendi\

Popis ostalih mogućih argumenata i način na koji se oni koriste možete vidjeti pomoću `help(plot)` naredbe.

Zadatak 1. 1.

Nacrtajte graf funkcije x^2 u intervalu $[-5, 5]$.

a. Dodajte naslov grafa i nazive osi.

b. Promijenite boju linije, debljinu i izgled linije prema želji. (Boju linije možemo promijeniti na više načina - pokušajte promijeniti boju na različite načine).

RJEŠENJE ZADATKA

```
plot(x^2,(-5,5),color='purple',thickness='2', linestyle= '--', title=('$x^{2}$'), axes_labels=(['$x$','$f(x)$']))
```

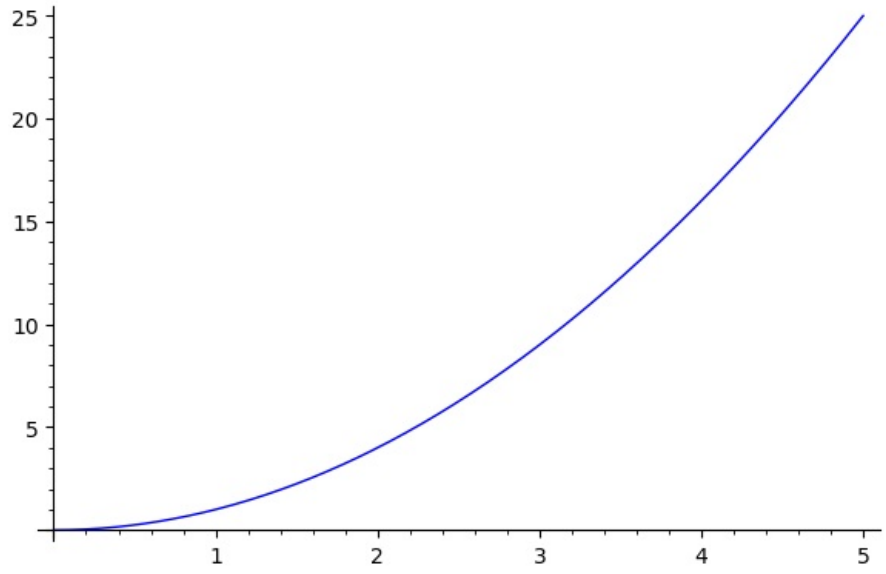
Grafove možemo crtati i tako da grafičkom objektu pridružimo neku varijablu koju onda kasnije možemo pozvati ili jednostavnije prikazati više funkcija na istom grafičkom prikazu. U sljedećem primjeru ćemo uz grafički prikaz kvadratne funkcije dodati i grafički prikaz funkcije x^3.

Primjer 1.2.

In [2]:

```
P_kvadrat = plot(x^2, (x,0,5))
P_kub = plot(x^3, (x,0,5))
P_kvadrat
```

Out[2]:

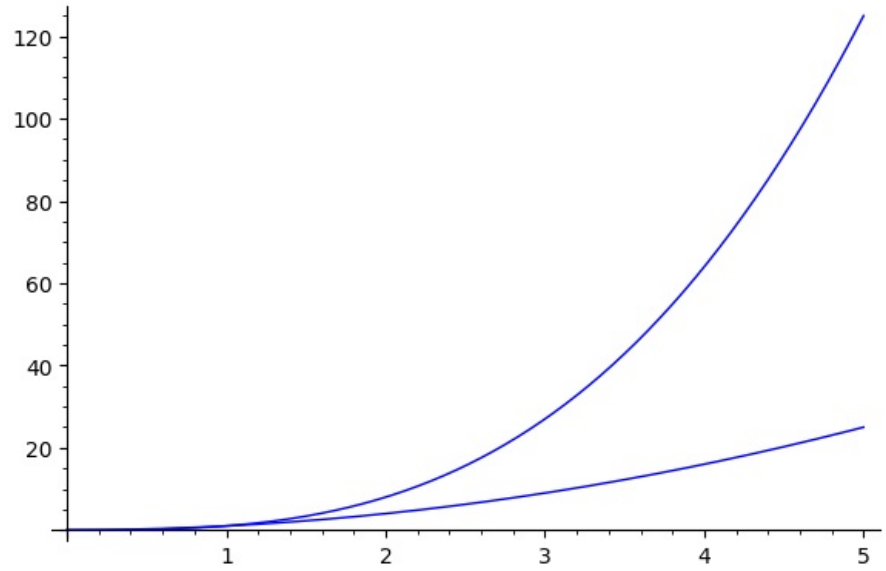


Obje funkcije prikazujemo na sljedeći način.

In [3]:

```
P_kvadrat + P_kub
```

Out[3]:

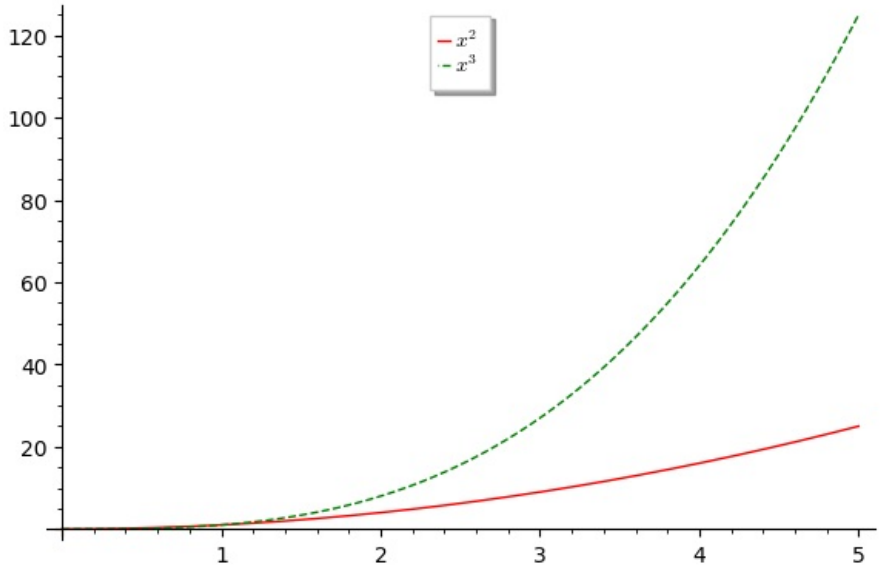


Prikazivanjem više različitih funkcija dolazimo do problema njihovog vizualnog razlikovanja i imenovanja. Razlikovati ih jednostavno možemo promjenom njihove reprezentacije (boje, debljine, izgleda,...), ali i dalje ostaje problem koja se linija odnosi na koju prikazanu funkciju. Na sljedećem primjeru ćemo prikazati kako možemo na graf dodati legendu.

In [4]:

```
P_kvadrat = plot(x^2, (x,0,5), color = 'red', legend_label='$x^2$') #unutar funkcije plot() dodajemo argument pomoću kojega grafičkom prikazu pridružujemo ime
P_kub = plot(x^3, (x,0,5), color = 'green',  linestyle = '--', legend_label='$x^3$')
graf = P_kvadrat + P_kub # u novu varijablu spremamo grafičke prikaze koje želimo prikazati zajedno
graf.set_legend_options(loc='upper center') # definiramo argumente za legendu (u ovom slučaju položaj legende)
graf
```

Out[4]:



Svoj graf možete i spremiti na računalo u različitim formatima: PDF, png i jpg.PDF ili slika grafa se sprema u direktorij u kojem se nalazi vaša Jupyter bilježnica.

Grafove spremamo koristeći `.save('naziv_datoteke.format')` . Ako želimo spremiti graf koji prikazuje funkcije x^2 i x^3 u obliku .png to bismo napravili pozivajući naredbu:

```
graf.save('graf.png')
```

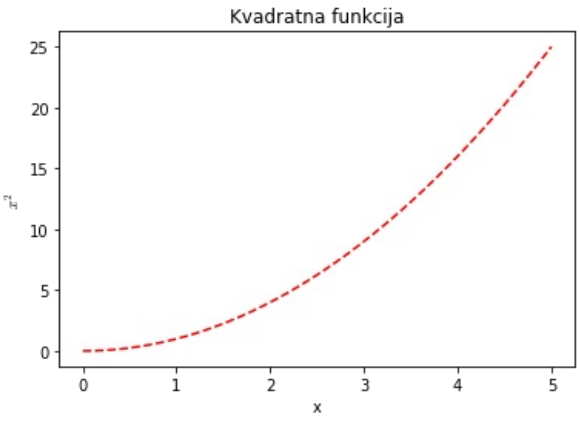
1.2. Crtanje grafova pomoću *Pythona*

Za one koji se bolje služe s Pythonom su se susreli s različitim modulima i knjižnicama koje se koriste unutar ovog programskog jezika. Kako je SageMath baziran na programskom jeziku Python onda su unutar njega integrirani i svi ti moduli i knjižnice. Tako da nam za crtanje grafova može pomoći i Phytonova knjižnica `Matplotlib` . Dokumentaciju za tu knjižnicu, kao i različite *tutoriale* možete pronaći na njenoj stranici (https://python-graph-gallery.com/matplotlib/). Unutar Phytona nam je za računanje vrijednosti funkcija i bilo kakva znanstvena računanja potreban i Pythonov paket `Numpy` . Njegovu dokumentaciju možete pronaći na sljedećoj stranici (https://numpy.org/).

Na sljedećem primjeru je prikazano kako neki graf možemo nacrtati koristeći `Matplotlib` knjižnicu.

In [5]:

```
import matplotlib.pyplot as plt # pristupanje potrebnim modulima
import numpy as np
x = np.linspace(0, 5, 10000) #definiranje raspona x koordinate (start, stop, broj generiranih uzoraka)
plt.plot(x,x^2, color='red', linestyle='--')
plt.title("Kvadratna funkcija")
plt.xlabel("x")
plt.ylabel("$x^{2}$");
```



Zadatak 1.2.

Znamo da srednju brzinu tijela možemo odrediti kao:

$$v = \frac{\Delta x}{\Delta t},$$

a kod jednolikog pravocrtnog gibanja je trenutna brzina jednaka srednjoj brzini. Znajući da je $\Delta x = x - x_0$ i $\Delta t = t - t_0$, možemo doći do sljedećeg izraza za ovisnost položaja o vremenu kod jednolikog pravocrtnog gibanja:

$$x = vt + x_0 - vt_0$$

.

Nacrtajte graf ovisnosti položaja tijela o vremenu za tijelo koje se giba jednoliko po pravcu za dva različita tijela za koja ćete sami odabrati vrijednosti v, t_0 i x_0. Navedite nazive osi (s mjernim jedinicama), na neki način prikažite razliku između prikaza gibanja (različita debljina, boja ili oblik linije) te dodajte legendu i pozicionirajte je tako da najbolje odgovara vašem grafu.

Zadatak napravite na dva načina:

1. korištenjem naredbi SageMatha i
2. korištenjem Pythonove knjižnice `Matplotlib` (potrebno je istražiti kako dodati legendu na graf u ovom slučaju).

RJEŠENJE ZADATKA

```
# SageMath naredbe
v = 10
x_0 = 10
t_0 = 10
var('t')
p1 = plot(v*t+x_0-v*t_0,(0,30), color='purple', linestyle= '--', axes_labels=(['$t/s$','$x/m$']), legend_l
abel='$v=10$ $m/s$, $x_{0}=10$ $m$, $t_{0}=10$ $s$')
v1 = 5
x_01 = 5
t_01 = 0
p2 = plot(v1*t+x_01-v*t_01,(0,30), color='orange', linestyle= '-', axes_labels=(['$t/s$','$x/m$']), legend
_label='$v=5$ $m/s$, $x_{0}=5$ $m$, $t_{0}=0$ $s$')
p1 + p2
```

```
# Pythonova knjižnica Matplotlib
import matplotlib.pyplot as plt
import numpy as np

t = var ('t')
t = np.linspace(0, 30, 100)
v = 10
x_0 = 10
t_0 = 10
plt.plot(t, v*t+x_0-v*t_0, color='purple', linestyle='--', label = '$v=10$ $m/s$, $x_{0}=10$ $m$, $t_{0}=1
0$ $s$')
v1 = 5
x_01 = 5
t_01 = 0
plt.plot(t, v1*t+x_01-v*t_01, color='orange', linestyle= '-', label = '$v=5$ $m/s$, $x_{0}=5$ $m$, $t_{0}=
0$ $s$')
plt.xlabel("$t/s$")
plt.ylabel("$x/m$")
plt.legend()
plt.show()
```

1.3. Crtanje grafova iz podataka

Kada u znanosti provodimo mjerenja ne dobivamo kontinuirane vrijednosti podataka kakve smo dosad prikazivali nego dobivamo točke u koordinatnom sustavu. Također, ako obrađujemo podatke mjerenja na računalu oni se mogu nalaziti spremljeni u datoteku. To je pogotovo slučaj kada se radi o velikoj količini podataka. U ovoj bilježnici ćemo vidjeti kako možemo dohvatiti ("pročitati") podatke spremljene u datoteci te kako dane podatke možemo grafički prikazati.

Kod razmjene podataka između različitih aplikacija i programa se najčešće koriste tekstualne CSV datoteke (*eng. Comma Separated Values*) ili tekstualne datoteke ekstenzije `.txt` .Unutar samog SageMath-a je čitanje takvih datoteka malo kompliciranije, ali kako je SageMath baziran na programskom jeziku Python njegove knjižnice i ugrađene funkcije nam olakšavaju manipulaciju CSV datotekama. Unutar Phyton-a postoje dvije knjižnice koje nam pomažu kod čitanja tekstualnih datoteka, kao i zapisivanja podataka u takve datoteke. Kada radimo s manjom količinom podataka (što ćemo najčešće i raditi) dovoljno nam je korisiti Pythonov `CSV modul` . S druge strane, ako imamo potrebu raditi s velikom količinom podataka ili numeričku analizu nad velikom količinom podataka onda možemo koristiti `Pandas knjižnicu` .

U ovoj bilježnici ćemo prikazati kako čitati datoteke koristeći `CSV modul` , a dokumentaciju za `Pandas knjižnicu` možete pronaći na sljedećoj stranici (https://pandas.pydata.org/).

Čitanje CSV datoteka

Recimo da želimo pročitati datoteku `SIsustav.csv` koju ste dobili uz ovu bilježnicu. Unutar datoteke se nalaze osnovne veličine SI sustava, njihove oznake, pripadne mjerne jedinice i oznaka tih mjernih jedinica. Za najjednostavnije čitanje datoteke, ta datoteka mora biti spremljena u isti direktorij kao i bilježnica u kojoj otvaramo željenu datoteku. U protivnom, uz ime datoteke moramo navesti i put do nje.

Najjednostavniji način za čitanje neke datoteke je prikazan na sljedećem primjeru:

In [6]:

```
SIsustav = open('SIsustav.csv', 'r') #otvara datoteku "SIsustav.csv" i sprema podatke unutar parametra SIsustav
for red in SIsustav:
    print (red)
SIsustav.close() #zatvara datoteku "SIsustav.csv"
```

```
fizikalna velicina, oznaka fizikalne velicine, mjerna jedinica, oznaka mjerne jedinice

duljina, l, metar, m

masa, m, kilogram, k

vrijeme, t, sekunda, s

temperatura, T, kelvin, K

kolicina tvari, n, mol, mol

jakost elektricne struje, I, amper, A

jakost svjetlosnog izvora, I, kandela, cd
```

Također možemo pročitati i samo prvi red unutar datoteke pomoću funkcije `readline()` .

In [7]:

```
SIsustav = open('SIsustav.csv', 'r')
SIsustav1 = SIsustav.readlines()
nova_lista=[]
for red in SIsustav1:
    nova_lista.append(red.rstrip('\n'))
print(nova_lista)
SIsustav.close()
```

```
['fizikalna velicina, oznaka fizikalne velicine, mjerna jedinica, oznaka mjerne jedinice', 'duljina,
l, metar, m', 'masa, m, kilogram, k', 'vrijeme, t, sekunda, s', 'temperatura, T, kelvin, K', 'kolici
na tvari, n, mol, mol', 'jakost elektricne struje, I, amper, A', 'jakost svjetlosnog izvora, I, kand
ela, cd']
```

In [8]:

```
SIsustav = open('SIsustav.csv', 'r')
lista_SIsustav = [] #stvaramo praznu listu u koju ćemo spremiti podatke iz datoteke "SIsustav.csv"
for red in SIsustav:
    lista_SIsustav.append(red.strip()) #pomoću for petlje dodajemo pročitane linije iz datoteke u listu, a korišt
enjem red.strip() uklanjamo oznaku za novi red (eng. newline) \n
SIsustav.close()

print(lista_SIsustav)
```

```
['fizikalna velicina, oznaka fizikalne velicine, mjerna jedinica, oznaka mjerne jedinice', 'duljina,
l, metar, m', 'masa, m, kilogram, k', 'vrijeme, t, sekunda, s', 'temperatura, T, kelvin, K', 'kolici
na tvari, n, mol, mol', 'jakost elektricne struje, I, amper, A', 'jakost svjetlosnog izvora, I, kand
ela, cd']
```

Gore prikazan način će dobro funkcionirati za datoteke u kojima se nalazi malo podataka. Čim imamo datoteku s velikom količinom podataka ovaj način čitanja datoteke će zauzimati puno memorije pa ga je bolje izbjegavati. Jednostavniji način rada i elegantniji način za maipulaciju podacima je preko Pythonovog `CSV modula` . Ako radimo u Pythonu onda prije svega moramo uvesti `CSV modul` naredbom:

```
import csv
```

i tek tada možemo pristupiti njegovim opcijama. Pogledajmo čitanje te datoteke i neke mogućnosti na primjeru.

In [9]:

```
import csv

with open('SIsustav.csv', 'r') as datoteka: #otvara datoteku za čitanje (i nakon završavanja zatvara)
    SIsustav = csv.reader(datoteka)
    for red in SIsustav:
        print(red)
```

```
['fizikalna velicina', ' oznaka fizikalne velicine', ' mjerna jedinica', ' oznaka mjerne jedinice']
['duljina', ' l', ' metar', ' m']
['masa', ' m', ' kilogram', ' k']
['vrijeme', ' t', ' sekunda', ' s']
['temperatura', ' T', ' kelvin', ' K']
['kolicina tvari', ' n', ' mol', ' mol']
['jakost elektricne struje', ' I', ' amper', ' A']
['jakost svjetlosnog izvora', ' I', ' kandela', ' cd']
```

Kao što možemo vidjeti na ovom primjeru, vrijednosti su već spremljene u liste po linijama pa možemo i jednostavnije pristupiti pojedinoj liniji ili stupcu. Također je prednost ovog načina što se datoteka odmah zatvara nakon čitanja.

In [10]:

```
import csv

with open('SIsustav.csv', 'r') as datoteka:
    SIsustav = csv.reader(datoteka)
    for red in SIsustav:
        print(red[0])#ispisuje vrijednosti koje se nalaze u svakoj liniji na poziciji 1
```

```
fizikalna velicina
duljina
masa
vrijeme
temperatura
kolicina tvari
jakost elektricne struje
jakost svjetlosnog izvora
```

Kako ne bismo stalno trebali otvarati datoteku i čitati je kada želimo manipulirati podacima koji su zapisani u njoj, korisno je podatke nakon čitanja spremiti u listu. Na taj način možemo pristupiti svim podacima i kada je datoteka zatvorena.

In [11]:

```
import csv

with open('SIsustav.csv') as datoteka:
    SIsustav = list(csv.reader(datoteka)) #čita datoteku i sprema pročitane podatke u listu

print(SIsustav)
```

```
[['fizikalna velicina', ' oznaka fizikalne velicine', ' mjerna jedinica', ' oznaka mjerne jedinice']
, ['duljina', ' l', ' metar', ' m'], ['masa', ' m', ' kilogram', ' k'], ['vrijeme', ' t', ' sekunda'
, ' s'], ['temperatura', ' T', ' kelvin', ' K'], ['kolicina tvari', ' n', ' mol', ' mol'], ['jakost
elektricne struje', ' I', ' amper', ' A'], ['jakost svjetlosnog izvora', ' I', ' kandela', ' cd']]
```

Sada smo dobili listu čiji su elementi liste koje sadrže podatke iz pojedine linije. Sada možemo jednostavno pristupiti pojedinom elementu liste unutar liste pozivanjem:

```
imeListe[x][y]
```

gdje x predstavlja listu kojoj želimo pistupiti unutar liste `imeListe` , a y element liste kojem želimo pristupiti. Pogledajmo to na primjeru liste `SIsustav` u koju su spremljeni podaci iz datoteke `SIsustav.csv` .

In [12]:

```
import csv

with open('SIsustav.csv') as datoteka:
    SIsustav = list(csv.reader(datoteka))

print(SIsustav[1][0]) #pristupamo 1. elementu u 1. liniji
print(SIsustav[1][1]) #pristupamo 2. elementu u 2. liniji
```

```
duljina
 l
```

ZADATAK 1.3.

Preuzmite na svoje računalo datoteku `mjerenja.csv` u kojoj se nalaze podaci mjerenja ovisnosti akceleracije o masi tijela pri djelovanju stalne sile od 2 N (spremite datoteku u isti direktorij gdje ćete spremati bilježnicu u kojoj radite). Mjerenje je provedeno za četiri različite mase tijela i za svaku masu su izmjerene četiri vrijednosti akceleracije.

1. Pročitajte datoteku `mjerenja.csv` i pročitane podatke spremite u listu.
2. Koristeći `for petlju` ili `list comprehension` razdvojite svaku liniju tako da dobijete 5 lista u kojima će se nalaziti svaka linija posebno.
3. Pronađite kako možete maknuti prvi element liste tako da u svakoj listi imamo samo izmjerene vrijednosti i primijenite to na svaku listu.
4. Podaci su unutar liste spremljeni u obliku stringa, a nama je za računanje potrebna njihova numerička vrijednost. Odredite numeričku vrijednost za svaki element svake liste i dobivene vrijednosti spremite u novu ili istu listu.

RJEŠENJE ZADATKA

```
import csv

with open('mjerenja.csv') as vrijednosti:
    mjerenja = list(csv.reader(vrijednosti))

for a in mjerenja:
    broj_elemenata = len(a)

masa = [float(mjerenja[0][i]) for i in range(1, broj_elemenata)]
a1 = [float(mjerenja[1][i]) for i in range(1, broj_elemenata)]
a2 = [float(mjerenja[2][i]) for i in range(1, broj_elemenata)]
a3 = [float(mjerenja[3][i]) for i in range(1, broj_elemenata)]
a4 = [float(mjerenja[4][i]) for i in range(1, broj_elemenata)]

print(masa)
print(a1)
```

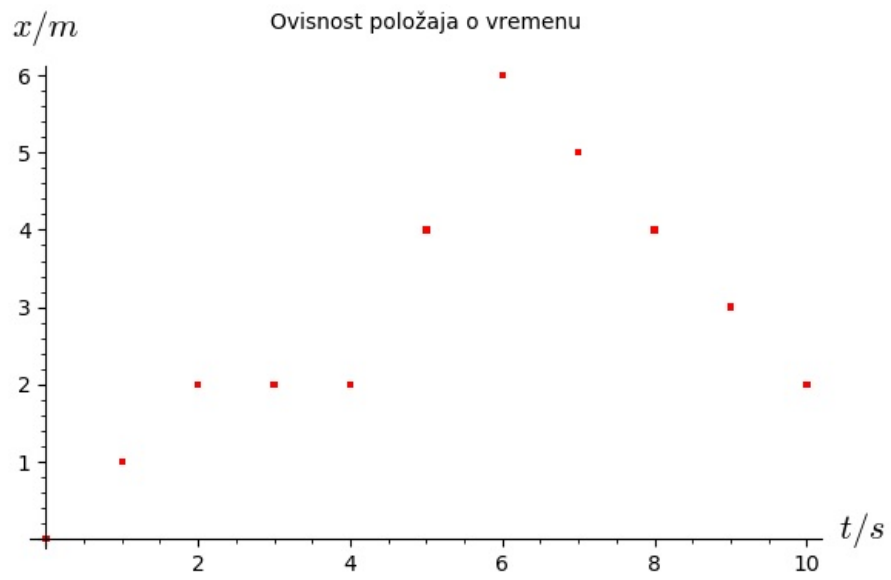
Crtanje grafova iz raspršenih podataka

Kao što smo ranije napomenuli, podaci koje dobivamo znanstvenim mjerenjima nisu kontinuirane, već raspršene vrijednosti. Zato je važno znati nacrtati graf iz raspršenih podataka. Ovdje je prikazan primjer gdje je potrebno nacrtati $x - t$ graf iz podataka spremljenih u liste koristeći ugrađene funkcije *SageMath*a i *Python*a.

In [13]:

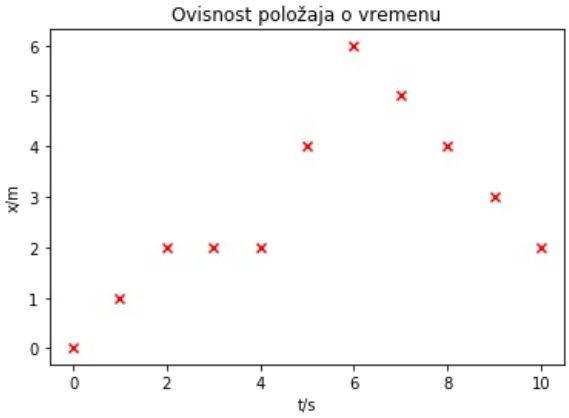
```
#primjer u kojem se koriste funkcije *SageMath*a za crtanje grafova iz raspršenih podataka
t = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x = [0, 1, 2, 2, 2, 4, 6, 5, 4, 3, 2]

x_t = list(zip(t,x)) #vraća listu tuple-ova koji u ovom slučaju predstavljaju koordinate

list_plot(x_t, color = 'red', title = 'Ovisnost položaja o vremenu', axes_labels = ['$t/s$','$x/m$'], marker = "s
")
#crta graf iz liste pri čemu smo dodali naziv grafa i nazive osi te promijenili boju i oblik makera
```

Out[13]:



In [14]:

```
#primjer u kojem se koriste funkcije *Python*a za crtanje grafova iz raspršenih podataka
import matplotlib.pyplot as plt # pristupanje potrebnom modulu

t = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x = [0, 1, 2, 2, 2, 4, 6, 5, 4, 3, 2]

plt.scatter(t,x, c = 'red', marker = 'x')
plt.title("Ovisnost položaja o vremenu")
plt.xlabel("t/s")
plt.ylabel("x/m")
plt.show;
```



Za više mogućih parametara pogledajte sljedeći link (https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.scatter.html).

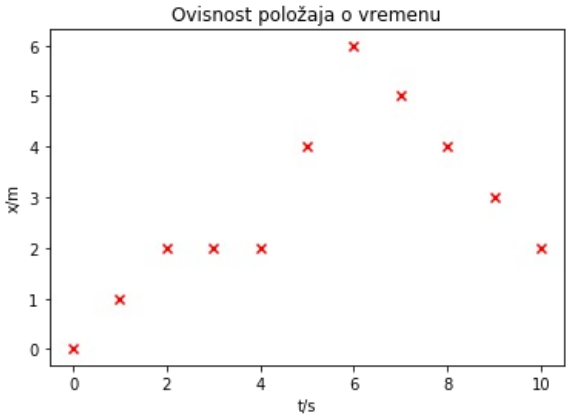
Na isti način možeo nacrtati i grafički prikaz kada su podaci spremljeni u listu *tuple-ova* kao što je to bio slučaju u prvom primjeru crtanja grafa. Da bismo mogli nacrtati graf iz liste *tuple-ova* moramo prvo "otpakirati" tu listu i taj zip u kojem su spremljeni podaci. To smo napravili pomoću operatora otpakiravanja * koji postavljamo prije objekta koji želimo otpakirati. On otpakirava vrijednosti bilo kojeg iterabilnog objekta. Za više detalja o operatoru otpakiravanja i kako ga možemo korisititi posjetite link (https://realpython.com/python-kwargs-and-args/#unpacking-with-the-asterisk-operators).

In [15]:

```
import matplotlib.pyplot as plt

t = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x = [0, 1, 2, 2, 2, 4, 6, 5, 4, 3, 2]

x_t = list(zip(t,x))

plt.scatter(*zip(*x_t), c = 'red', marker = 'x')
plt.title("Ovisnost položaja o vremenu")
plt.xlabel("t/s")
plt.ylabel("x/m")
plt.show
```

Out[15]:

```
<function show at 0x6fe47515d90>
```



ZADATAK 1.4.

Nakon rješavanja 1. zadatka trebali biste imati 5 lista u kojima se nalaze numeričke vrijednosti izmjerenih podataka za masu tijela i za vrijednosti akceleracija izmjerenih za svaku masu.

1. Izračunajte srednje vrijednosti akceleracija izmjerenih za svaku pojedinu masu i dobivene srednje vrijednosti spremite u novu listu. Nova lista treba imati jednak broj elemenata kao i lista u kojoj su spremljene vrijednosti masa.
2. Nacrtajte graf ovisnosti akceleracije o masi tijela. Podaci na grafu ne smiju biti povezani linijama. Graf obavezno treba sadržavati naziv grafa i nazive osi s pripadnim mjernim jedinicama.
3. Za vrijednosti masa u rasponu masa koje su dane mjerenjima odredite teorijske vrijednosti akceleracija preko II. Newtonovog zakona ako znate da je na tijela djelovala stalna sila od 2 N. Dobivene vrijednosti spremite u novu listu.
4. Na istom grafu kao i ranije dodajte ovisnost akceleracije o masi za teorijske vrijednosti. Teorijsku krivulju prikažite punom linijom. Na graf dodajte i legendu.

RJEŠENJE ZADATKA

```
import matplotlib.pyplot as plt
import numpy as np

prosjek = [sum(a1)/len(a1), sum(a2)/len(a2), sum(a3)/len(a3), sum(a4)/len(a4)]
masa_raspon = np.linspace(masa[0], masa[3], 20)

plt.scatter(prosjek,masa, c = 'red', marker = 'x')
plt.plot(2/masa_raspon, masa_raspon)
plt.title("Ovisnost akceleracije o masi")
plt.xlabel("m[kg]")
plt.ylabel("a[$m/s^2$]")
plt.legend(["mjerenja", "teorija"], loc ='upper right')
plt.show()
```

Poglavlje 2: Numeričke metode određivanja nultočaka funkcije

Koliko su nam grafički prikazi u znanosti bitni za jednostavniji prikaz određenog problema, toliko je i bitno znati analizirati dobivene grafove. Analizom možemo detaljnije opisati, a možda i pronaći rješenje našeg problema. Svi grafovi prikazuju nekakvu funkciju (ovisnost jedne varijable o drugoj) pa je analiza grafa zapravo analiza funkcije koju taj graf prikazuje. Najosnovnije sastavnice analize funkcija su pronalaženje nultočaka funkcije i njenih ekstrema (minimuma i maksimuma).

2.1. Općenito o nultočkama funkcije

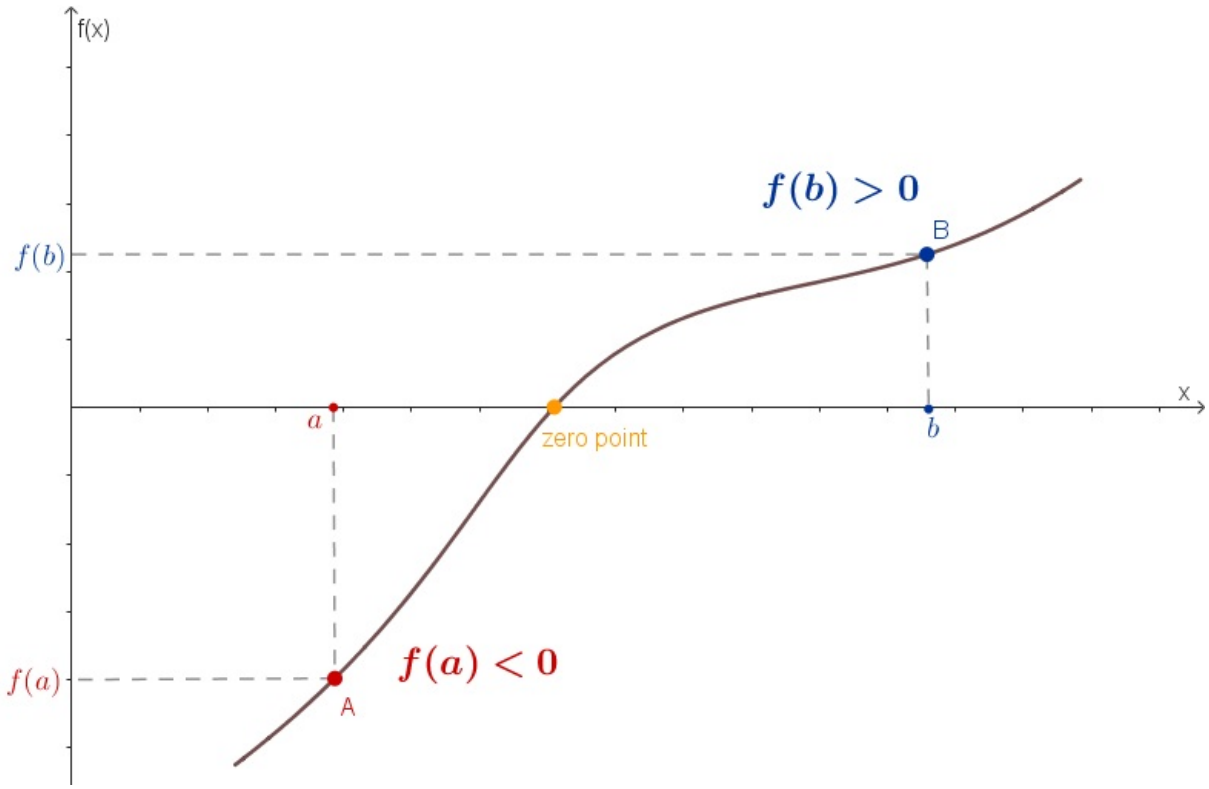
Nultočke funkcije su točke u kojima dana funkcija siječe os apscisu, a dobivamo ih tako da funkciju izjednačimo s nulom. Većina funkcija s kojima ste se dosad susretali tijekom svog školovanja su **polinomi**, funkcije oblika $a_n x^n + a_{n-1}x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$ gdje $n \in \mathrm{N}$ predstavlja stupanj polinoma. Kada se radi o polinomu niskog stupnja ($n = 1, 2$) određivanje nultočaka funkcije nije problem. Ako naprimjer imamo funkciju $f(x) = 2x + 4$, postupak određivanja nultočke funkcije je vrlo jednostavan: $2x + 4 = 0 \rightarrow 2x = -4 \rightarrow x = -2$. Kod polinoma drugog stupnja (kvadratna funkcija), nultočke funkcije $f(x) = ax^2 + bx + c$ dobivamo rješavanjem kvadratne jednadžbe. A rješenje kvadratne jednadžbe je općenito određeno, dobro poznatim izrazom, $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

Slični općeniti, iako kompliciraniji izrazi postoje i za polinome trećeg i četvortog stupnja. No, za polinome petog (pa i višeg) stupnja ne postoji takvo općenito rješenje. Također, nisu ni funkcije samo polinomi. Postoje i trigonometrijske funkcije, eksponencijalne funkcije, ili funkcije koje su kombinacija ranije navedenih funkcija. I za mnoge od tih funkcija ne postoji analitičko rješenje (rješenje zapisano preko elementarnih funkcija) ili ako postoji, ono je jako komplicirano. To naravno nije slučaj samo s pronalaženjem nultočaka funkcije, nego taj problem postoji i kod drugih elemenata analize funkcija i njihovih grafičkih prikaza. Tu se onda javlja potreba za numeričkom analizom, odnosno pronalaženjem zadovoljavajućeg numeričkog rješenja.

U ovoj bilježnici ćemo opisati dvije numeričke metode pomoću kojih se mogu odrediti nultočke funkcije: **metoda bisekcije** i **metoda sekante**. Uz ove dvije postoje još neke numeričke metode pronalaženja nultočaka funkcije od kojih je najpoznatija *Newton-Raphsonova metoda*. Za razliku od metode bisekcije i metode sekante, kod Newton-Raphsonove metode je potrebno manje iteracija kako bismo došli do željene preciznosti, ali je mi nećemo ovdje prikazivati jer je za nju potrebno veće znanje matematike.

2.2. Metoda bisekcije

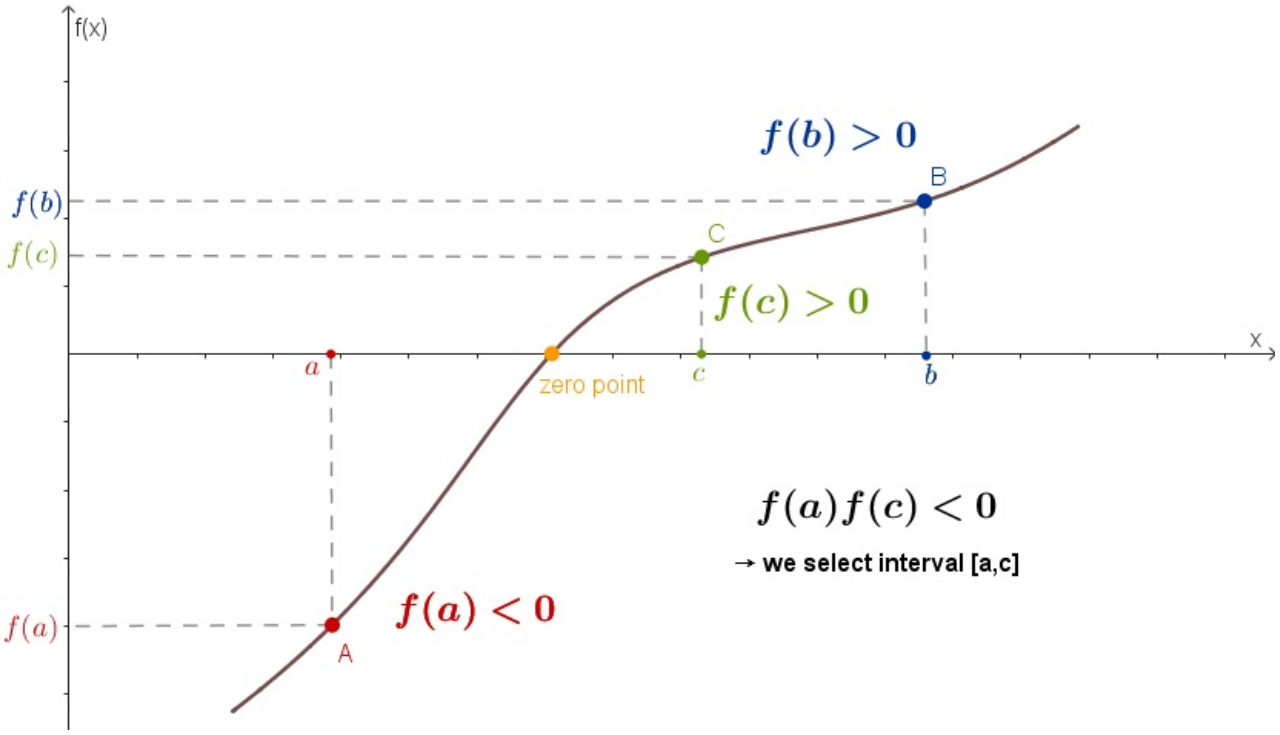
Osnovna ideja ove metode je vrlo jednostavna. Recimo da je poznato da se nultočka funkcije $f(x)$ nalazi u intervalu $[a, b]$. Kod funkcija koje možemo grafički prikazati, s grafičkog prikaza možemo odrediti taj interval. Ako je grafički prikaz funkcije kompliciran za odrediti, interval u kojem se nalazi nultočka funkcije možemo dobiti tako da pogledamo u kojim vrijednostima dolazi do promjene predznaka funkcije. U tom slučaju trebamo odabrati dvije proizvoljne vrijednosti $x = a$ i $x = b$ te odrediti njihovu vrijednost $f(a)$ i $f(b)$. Ako je $f(a) > 0$ (pozitivna), a $f(b) < 0$ (negativna) ili obrnuto onda se u intervalu $[a, b]$ nalazi nultočka jer je unutar tog intervala fukcija morala proći kroz os apscisu kako bi se promijenio predznak. Općeniti primjer je prikazan na slici pronalaženja odgovarajućeg intervala je prikazan na Slici 1.



Slika 1.
Prikaz intervala koji omeđuje nultočku.

Dakle, ako smo pronašli interval $[a, b]$ u kojem se nalazi nultočka funkcije $f(x)$ kod metode bisekcije je potrebno učiniti sljedeće:

1. Prepolovimo interval $[a, b]$ gdje je točka $c = \frac{a+b}{2}$ sredina tog intervala. Ako je $f(c) = 0$, onda je točka c nultočka pa smo gotovi s primjenom algoritma. U suprotnom, u jednom od intervala $[a, c]$ ili $[c, b]$ će se nalaziti nultočka.
2. Pronađemo koji od ta dva intervala sadrži nultočku tako da pogledamo usporedimo predznake funkcije u točki c i točki a.Informacija o odnosu predznaka funkcije u točki c i točki a (ili točki b) će nam otkriti kroz koji interval prolazi naša funkcija i s kojim intervalom dalje nastavljamo raditi. Ako je $f(c)f(a) > 0$ onda obje funkcije imaju isti prezdnak i u intervalu $[a, c]$ se ne nalazi nultočka (jer funkcija ne siječe os apscisu) pa onda odabiremo interval $[c, b]$. Ako je $f(c)f(a) < 0$ onda dolazi do promjene predznaka, odnosno funkcija siječe os apscisu u tom intervalu i u tom slučaju odabiremo interval $[a, c]$. Općeniti primjer možete vidjeti na Slici 2.



Slika 2.
Prikaz generalne ideje određivanja intervala u kojem se nalazi nultočka kod metode bisekcije.

1. Nastavljamo proces bisekcije dok ne dobijemo dovoljno malen interval u kojem se nalazi nultočka funkcije, odnosno dok ne dobijemo interval koji odgovara željenoj preciznosti određivanja nultočke. Interval postaje dovoljno malen onda kada je razlika između granica intervala manja ili jednaka odabranoj preciznosti, odnosno iteriramo dok vrijedi da je $|b-a| > preciznosti$. U svakoj od iteracija ponovno moramo znati u kojem se intervalu nalazi naša nultočka.

2.2.1. Kako konstruirati program koji provodi metodu bisekcije?

1. Definiramo funkciju koja nam prima 3 varijable: **a** - donju granicu početnog intervala, **b** - gornju granicu početnog intervala i **p** - preciznost. Tražimo sredinu početnog intervala, $c = \frac{a+b}{2}$: ako je ta točka nultočka ($f(c) = 0$) onda se program zaustavlja i vraća vrijednost nultočke, a ako nije provjeravamo predznake jednog od intervala (npr. je li $f(a)f(c) > 0$ ili je $f(a)f(c) < 0$). Ako je $f(a)f(c) < 0$ onda trebamo odabiremo interval $[a, c]$, odnosno postavljamo $a \rightarrow a$ i $b \rightarrow c$. Ako ne vrijedi taj uvjet onda postavljamo $a \rightarrow c$ i $b \rightarrow b$.Takve iteracije se ponavljaju dok ne zadovoljimo uvjet za preciznost, odnosno sve dok vrijedi da $|b-a| > p$. Konačno, funkcija vraća određenu nultočku.
2. Provjerimo jesu li predznaci funkcije u granicama početnog intervala suprotni, odnosno vrijedi li da je $f(a)f(b) < 0$. Ako nisu algoritam se zaustavlja, a ako jesu onda se provodi metoda bisekcije.

ZADATAK 2.1.

Napravite funkciju koja provodi metodu bisekcije. Kako biste isprobali kako ona radi, definirajte si neku poznatu funkciju (za koju znate nultočke) unutar te iste ćelije.

RJEŠENJE ZADATKA

```
def bisekcija (a, b, p):
    korak = 1
    uvjet = abs(b-a)
    while uvjet > p: #iteriramo dokle god vrijedi uvjet
        c = (a+b)/2.0 #tražimo sredinu početnog intervala
        print('Iteracija-%d, c = %0.6f i f(c) = %0.6f' % (korak, c, f(c)))
        if f(c) == 0.0: #provjeracamo je li točka c nultočka
            print ('Pronađena je nultočka funkcije u c=%0.6f i f(c)=%0.6f' %(c, f(c)))
            break
        elif f(a)*f(c) < 0.0: #ako točka c nije nultočka provjeravamo nalazi li se nultočka u intervalu a,
c
            b = c #ako uvjet vrijedi postavljam b=c, odnosno odabiremo interval a,c
        else:
            a = c #ako uvjet ne vrijedi onda postavljamo a=c, odnosno odabiremo interval b,c
        korak = korak + 1 #povećavamo korak
        uvjet = abs (b-a) #updateamo preciznost prema novom intervalu
    print ('\nNultočka funkcije je: %0.8f' %c)

#unos vrijednosti a, b i p
a = float(input('Donja granica početnog intervala: '))
b = float(input('Gornja granica početnog intervala: '))
p = float(input('Preciznost (unos s decimalnom točkom): '))

""" provjeravamo nalazi li se nultočka u početnom intervalu: ako se ne nalazi program javlja da odabremo n
ove
vrijednosti, a ako se nalazi onda se primjenjuje funkcija bisekcija(a,b,p) """
#primjer funkcije s poznatom nultočkom
f = x-2

if f(a)*f(b) >= 0.0:
        print ("\nUnutar tog intervala se ne nalazi nultočka!")
        print ("Pokušajte ponovno s novim vrijednostima.")
else:
    bisekcija (a, b, p)
```

Iako je metoda bisekcije jednostavna za shvatiti i implementirati, te uvijek konvergira prema nultočki pa je zbog toga pouzdana, postoje i nedostaci. Ona uvijek konvergira prema nultočki, ali konvergencija je jako spora, odnosno potrebno je puno iteracija kako bi se pronašla nultočka funkcije. I to samo vrijedi ako je funkcija kontinurana (ako ne postoje prekidi funkcije u tom intervalu) i ako je primenjujemo u točkama u kojima funkcija ima suprotne predznake. Zbog te potrebe, metodom bisekcije ne možemo pronaći nultočku za sve funkcije. Npr., metoda bisekcije nam neće pomoći ako želimo pronaći nultočku funkcije $f(x) = x^2$. Iako postoje brže i bolje numeričke metode za pronalaženje nultočki funkcije, s obzirom na naše ograničeno znanje matematike i za naše potrebe, metoda biskecije služi svrsi.

ZADATAK 2.2. - FIZIKALNI PROBLEM

Igrač ispuca loptu brzinom 5 m/s pod kutem od 40° u odnosu na tlo. Na kojoj udaljenosti od igrača će lopta udariti o tlo?

```
a. Definirajte funkciju putanje kosog hitca za uvjete navedene u zadatku.
b. Nacrtajte graf putanje kosog hitca kako biste okvirno mogli odrediti interval u kojem se nalazi nultočk
a te funkcije.
c. Primjeniti metodu bisekcije da pronađete domet lopte.
```

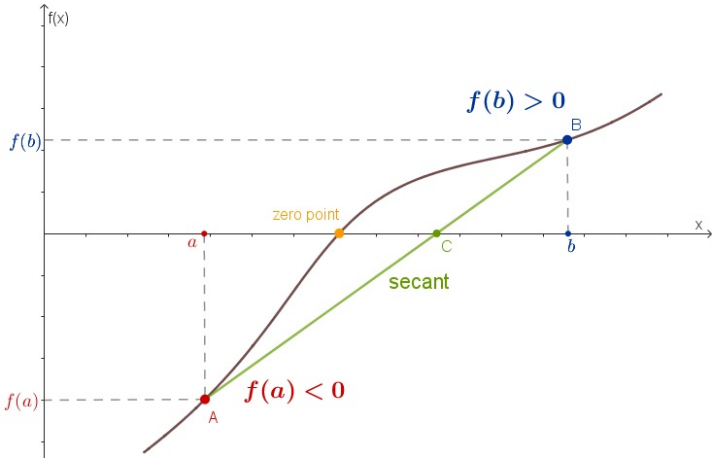
RJEŠENJE ZADATKA

```
import matplotlib.pyplot as plt
import numpy as np

g = 9.81
z = np.linspace (0,5,50)

v0 = float(input("Početna brzina (m/s): "))
alpha = float(input("Kut izbacivanja (°): "))

y = z*np.tan(alpha*np.pi/180)-(z^2 * g)/(2*v0^2*(np.cos(alpha*np.pi/180))^2)

plt.plot(z,y)
plt.xlabel ('x/m')
plt.ylabel ('y/m')
plt.grid(True)
plt.show()

def bisekcija (a, b, p):
    korak = 1
    uvjet = abs(b-a)
    while uvjet > p:
        c = (a+b)/2.0
        if f(c) == 0.0:
            break
        elif f(a)*f(c) < 0.0:
            b = c
        else:
            a = c
        korak = korak + 1
        uvjet = abs (b-a)
    print ('\nNultočka funkcije je: %0.8f' %c)

print ('Određivanje nultočke funkcije!')
a = float(input('Donja granica početnog intervala: '))
b = float(input('Gornja granica početnog intervala: '))
p = float(input('Preciznost (unos s decimalnom točkom): '))

f = x*np.tan(alpha*np.pi/180)-(x^2 * g)/(2*v0^2*(np.cos(alpha*np.pi/180))^2)


if f(a)*f(b) >= 0.0:
        print ("\nUnutar tog intervala se ne nalazi nultočka!")
        print ("Pokušajte ponovno s novim vrijednostima.")
else:
    bisekcija (a, b, p)
```

2.3. Metoda sekante

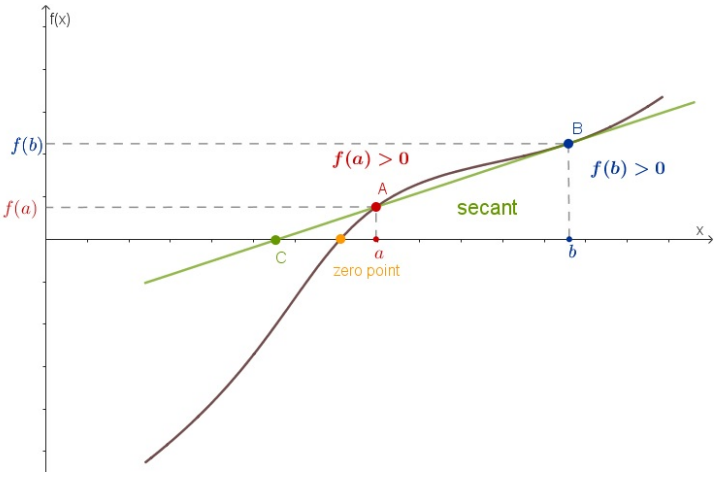
Metoda sekante je varijacije ranije spomenute Newton-Raphsonove metode, ali nam je za nju dovoljno osnovno znanje matematike. Isto kao i kod metode bisekcije i kod metode sekante počinjemo s dvije početne točke, odnosno s intervalom u kojem smatramo da se nalazi naša nultočka, ali je generalna ideja metode sekante malo dugačija. Jedna od prednosti ove metode, u odnosu na metodu bisekcije, je što funkcija u početnim točkama ne treba imati različite predznake (odoabrani interval na početku ne treba sadržavati nultočku).

Nakon odabira početnog intervala, tražimo sekantu funkcije u te dvije točke (pravac koji siječe funkciju u dvije točke). Princip je geometrijski prikazan na sljedećoj slikama 3 i 4. Na slici 3 je prikazan slučaj kada funkcija ima suprotne predznake u početnim točkama, a na slici 4 kada funkcija ima iste predznake u početnim točkama. U oba slučaja sekanta siječe os apscisu u nekoj točki c.



Slika 3.

Određivanje sekante u slučaju kada funkcija ima suprotne predznake u početnim točkama.



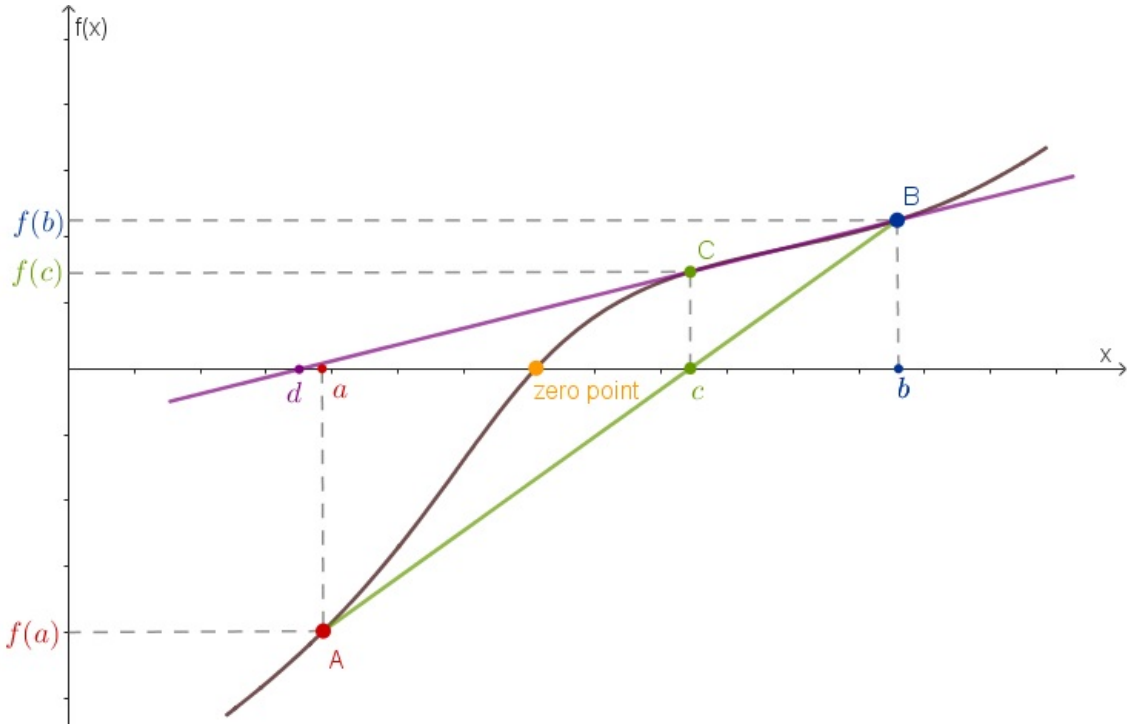
Slika 4.

Određivanje sekante u slučaju kada funkcija ima iste predznake u početnim točkama.

Da bismo odredili točku u kojoj sekanta siječe os apscisu, trebamo prvo odrediti jednadžbu sekante (jednadžbu pravca koji prolazi kroz točke a i b). Izjednačavanjem jednadžbe sekante s nulom, dobivamo poziciju točke c. Točnije, točku c možemo izračunati kao:

$$c = b - \frac{(b-a)f(b)}{f(b) - f(a)}$$

Nakon što odredimo točku u kojoj sekanta siječe os apscisu ponavljamo postupak za tu točku i točku b, kao što je prikazano na slici 5. Nova sekanta je na slici označena ljubičastom bojom i možemo vidjeti da se njen nagib mijenja tako da se sve više približava nagibu tangente u nultočki funkcije.



Slika 5.

Traženje druge sekante (označena ljubičastom bojom) preko novih točaka.

S iteracijama se zaustavljamo, kao i kod metode bisekcije, kada dođemo do zadovoljavajuće preciznosti (dokle god je zadovoljen uvjet $|f(c)| > p$.

Za razliku od metode biskecije, ovdje smo vidjeli da ne trebamo imati početan interval takav da funkcija u rubnim vrijednostima intervala ima suprotne predznake. Još jedna pozitivna stvar kod metode sekante je što ćemo doći do rješenje za nultočku nakon manjeg broja ponavljanja jer je konvergencija prema nultočki brža nego kod metode bisekcije. No, veliki nedostatak metode sekante je što se može dogoditi da nemamo konvergenciju prema nultočki funkcije i što moramo paziti da nam vrijednosti funkcije u točkama pomoću kojih određujemo sekantu nisu jednake jer u tome slučaju sekanta ne siječe os apscisu. Ipak, i uz te nedostatke, metoda sekante je dobra opcija kod određivanja nultočke u fizikalnim problemima gdje se uglavnom pojavljuju kontinuirane neosclirajuće funkcije.

2.3.1. Algoritam za izvođenje metode sekante

1. Definiraj f(x).
2. Definiraj ulazne vrijednosti: donju i gornju granicu intervala (a, b), preciznost (p) i maksimalni broj koraka (N). NAPOMENA: Dok su a, b i p *float* vrijednosti, N mora biti *integer*.

3. Inicijaliziraj brojač koraka: $i = 1$.
4. Ako je $f(a) = f(b)$ ispiši "Pogreška: nemoguće je dijeliti s nulom." i idi na (10), u suprotnom idi na (5).
5. Izračunaj $c = b - \frac{(b-a)f(b)}{f(b)-f(a)}$.
6. Povećaj broj koraka: $i = i + 1$.
7. Ako je $i \geq N$ ispiši "Ne konvergira." i idi na (10), u suprotnom idi na (8).
8. Ako je $|f(c)| > p$ postavi $a = b$ i $b = c$ i idi na (5), u suprotnom idi na (9).
9. Ispiši korijen kao c.
10. Zaustavi program.

ZADATAK 2.3.

Na temelju navedenog algoritma napišite funkciju koja će određivati nultočku funkcije pomoću metode sekante i provjerite njenu ispravnost na funkcijama čije ste nultočke tražili preko metode bisekcije.

RJEŠENJE ZADATKA

```
def sekanta (a, b, p, N):
    korak = 1
    uvjet = True
    while uvjet > p:
        if f(a)==f(b):
            print ('Pogreška! Nemoguće dijeliti s nulom.')
            break
        c = b -((b-a)*f(b))/(f(b)-f(a))
        print('\nIteracija-%d, c = %0.6f i f(c) = %0.6f' % (korak, c, f(c)))
        a = b
        b = c
        korak = korak + 1
        if korak > N:
            print ('Ne konvergira!')
            break
        uvjet = abs (f(c))
    print ('\nNultočka funkcije je: %0.8f' %c)

#unos vrijednosti a, b i p
a = float(input('Donja granica početnog intervala: '))
b = float(input('Gornja granica početnog intervala: '))
p = float(input('Preciznost (unos s decimalnom točkom): '))
N = int(input('Maksimalan broj ponavljanja: '))


f = x - 2

if f(a)==f(b):
    print ('Pogreška! Nemoguće dijeliti s nulom.')
    print ('Pokušajte ponovno s novim vrijednostima.')
else:
    sekanta(a, b, p, N)
```

ZADATAK 2.4.

Istražite dokumentaciju za *Python*ovu ugrađenu funkciju *fsolve* i iskoristite je za određivanje nultočaka funkcije $f(x) = x^3 + 2x - 1$.

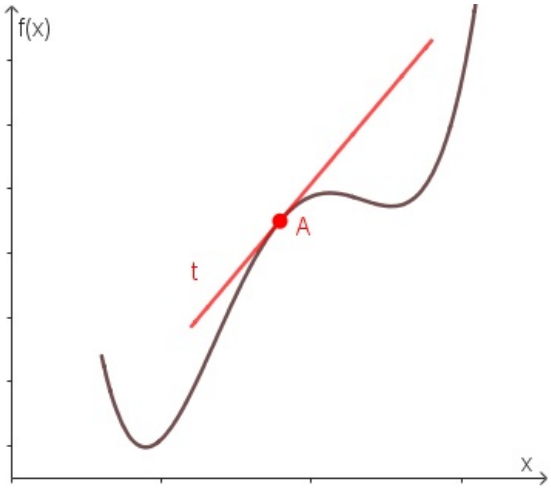
RJEŠENJE ZADATKA

```
import numpy as np
from scipy.optimize import fsolve
def func(x):
    return x^3 +2*x -1
root = fsolve(func,0)
root
```

Poglavlje 4: Numerička derivacija

Jedan od čestih problema u fizici prilikom analize nekog grafičkog prikaza je određivanje tangente funkcije u nekoj točki. Tangenta u nekoj točki funkcije nam donosi mnoge korisne informacije kod analize problema. U x-t grafu nagib tangante nam donosi infirmaciju o brzini tijela ili u v-t grafu o akceleraciji tijela u nekom određenom trenutku. Ovo su možda i najpoznatiji primjeri iz fizike gdje nam je poznavanje nagiba tangente korisno za analizu nekog

problema, ali nisu jedini. Iz nagiba tangente se općenito može saznati brzina promjene prikazane funkcije u odnosu na promjenu nezavisne varijable. To naravno nije primjenjivo samo u fizici nego u gotovo svim prirodnim znanostima, a i nekim društevnim znanostima poput ekonomije. Rješenje problema koje nam geometrijski donosi tangenta funkcije u nekoj točki funkcije, analitički nam predstavlja smanjenje intervala u kojem promatramo neku promjenu funkcije do trenutka kada je taj interval dovoljno mali da možemo odrediti vrijednost funkcije u nekoj točki. Jedan od takvih primjera u fizici je određivanje trenutne brzine i trenutne akceleracije nekog tijela tijekom njegovog gibanja što geometrijski možemo odrediti preko tangente funkcije u odeđenom trenutku gibanja. Jedan općeniti primjer određivanja tangente u točki T funkcije je prikazan na slici 1. Određivanje jednažbe tangente pa i njene vrijednosti u promatranoj točki funkcije nam omogućuju **derivacije**.

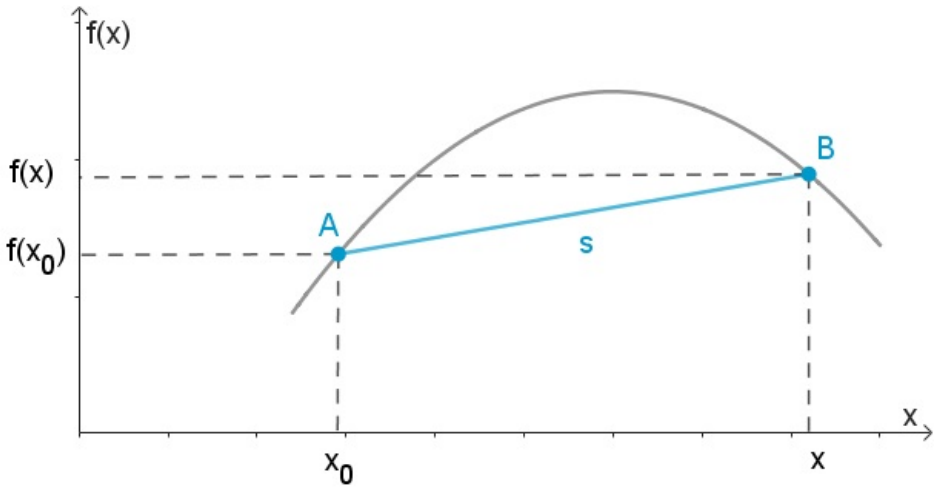


Slika 1.
Pravac p je tangenta funkcije u točki T, a derivacija funkcije odgovara nagibu pravca p u toj točki.

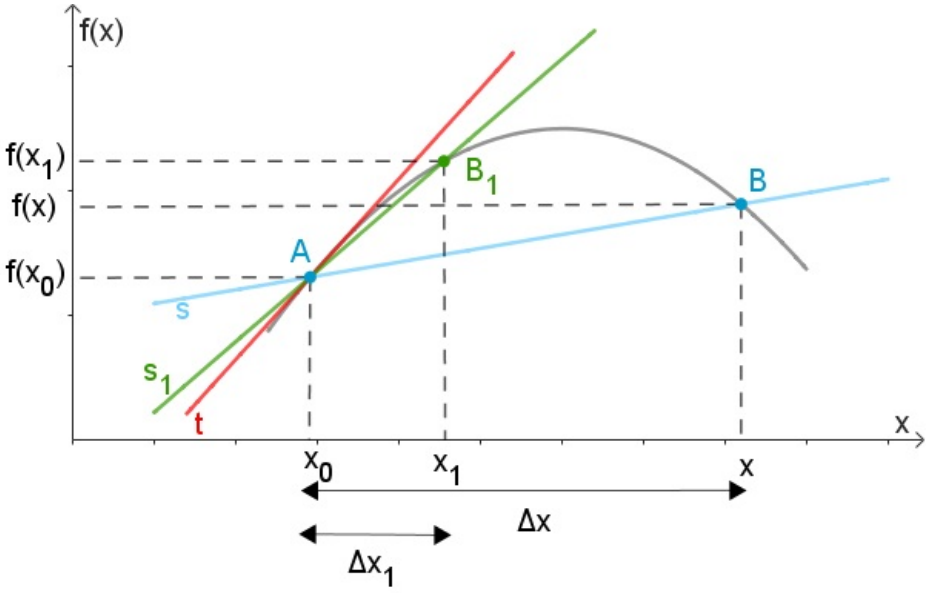
Iako trenutno nemamo dovoljno matematičkog znanja za određivanje derivacija funkcija, one se analitički mogu odrediti kod mnogih funkcija. No, opet postoje funkcije kod kojih analitičko određivanje derivacije nije jednostavno ili nije moguće. Tu nam ponovno koriste numeričke metode pa ćemo se u ovoj bilježnici detaljnije baviti određivanjem derivacije funkcije u nekoj točki funkcije.

4.1. Metoda konačnih razlika

Do pojma derivacije funkcije su istovremeno i nezavisno jedan o drugome došla dva znanstvenika: Isaac Newton i Gottfied Wilhelm Leibniz. Newton je proučavao brzinu tijela u danom trenutku, a Leibniz se bavio pitanjem tangente funkcije u nekoj točki. Već smo spomenuli da derivacija predstavlja brzinu promjene funkcije u odnosu na promjenu nezavisne varijable, odnosno geometrijski se derivacija funkcije u točki može interpretirati kao nagib tangente u toj točki funkcije. Najjednostavnije numeričko određivanje derivacije funkcije u nekoj točki funkcionira na istom principu na kojem je Leibniz došao do pojma derivacije, promatranjem sekante na funkciju (slika 2) i smanjivanjem intervala između dvije točke u kojima sekanta siječe graf funkcije. Na slici 3 možemo vidjeti da se smanjivanjem promatranog intervala Δx nagib senakte približava nagibu tangente u točki T_0.



Slika 2.
Sekanta siječe prikazanu funkciju u točkama T_0 i T.



Slika 3.
Smanjivanjem intervala Δx nagib senkate se približava nagibu tangente u točki T_0

Nagib (koeficijent smjera) sekante numerički možemo odrediti kao:

$$k_s = \frac{f(x) - f(x_0)}{x - x_0},$$

gdje $\Delta x = x - x_0$ predstavlja **promjenu varijable**, a $f(x) - f(x_0)$ **promjenu funkcijske vrijednosti varijable**. Samnjivanjem promjene varijable, odnosno kada $\Delta x \to 0$, sekanta prelazi u tangentu u točki x_0 kako je prikazano na slici 3. Na temelju toga, derivaciju funkcije $f'(x)$ u nekoj točki x_0 možemo zapisati kao

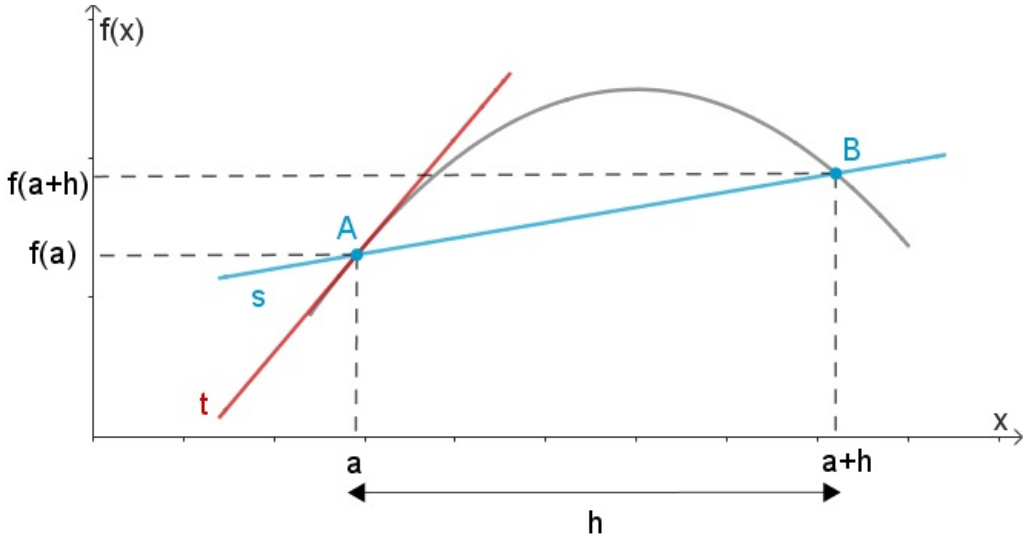
$$f'(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}.$$

Na temelju prethodnog izraza je vidljivo da za dovoljno male razmake između točaka, odnosno dovoljno mali Δx nagib sekate k_s je dovoljno dobra aproksimacija derivacije funkcije. Ukoliko odaberemo premalen korak Δx pojavljuje se greška zaokruživanja - prilikom oduzimanja članova u nazivniku dolazi do njihovog poništavanja i dobivamo nulu. S druge strane, ako korak Δx bude prevelik, nagib sekante nam neće biti dovoljno dobra aproksimacija derivacije (nagiba tangente) [Škec, L., Mujaković, N. & Dražić, I. (2010) Numerička analiza aproksimacija derivacije metodom konačnih razlika. Zbornik radova Građevinskog fakulteta Sveučilišta u Rijeci, 12, 145-168.]. Opisani način aproksimacije derivacije funkcije u nekoj točki nazivamo **metodom konačnih razlika**.

Radi jednostavnosti analize ćemo promatrati derivaciju funkciju u točki a gdje je duljina intervala između dvije promatrane točke h. Odnosno, kada postavimo da je $x_0 \to a$ i $\Delta x \to h$ uz pretpostavku da je $h > 0$ izraz za nagib sekante k_s koji ćemo koristiti za aproksimaciju derivacije $f'(a)$ postaje:

$$f'(a) \approx \frac{f(a + h) - f(a)}{h}.$$

Opisana zamjena varijabli je grafički prikazana na slici 4. Taj izraz predstavlja formulu za **aproksimaciju derivacije konačnim razlikama unaprijed (eng. *forward difference*)**. Takav izraz se dobiva iz razloga što se, prema slici 4, okolna točka odabire desno od točke u kojoj tražimo derivaciju funkcije (u ovom slučaju je to točka $(a, f(a))$).

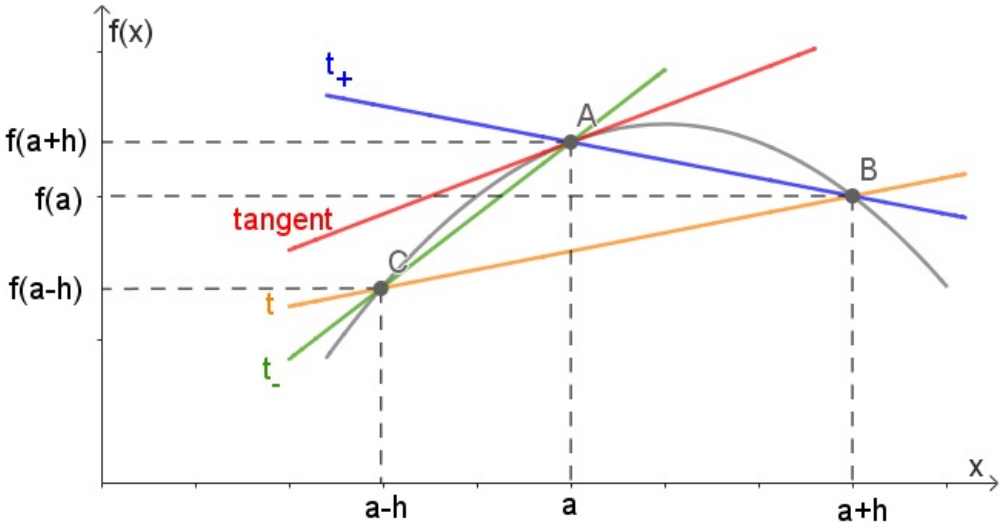


Slika 4.
Smanjivanjem udaljenosti između dvije točke, h, nagib senkate se približava nagibu tangente u točki T_0.

Do sličnog izraza kao za aproksimaciju derivacije konačnim razlikama unaprijed možemo dobiti ako okolnu točku odaberemo lijevo od točke u kojoj želimo odrediti derivaciju. U tom slučaju, koordinate okolne točke su $(a - h, f(a - h))$ pa je izraz za aproksimaciju derivacije dan na sljedeći način

$$f'(a) \approx \frac{f(a) - f(a-h)}{h}.$$

i predstavlja formulu za **aproksimaciju derivacije konačnim razlikama unazad (eng. *backward difference*)**.



Slika 5.
Grafička interpretacija formule za aproksimaciju derivacije konačnim razlikama unaprijed (plava), unazad (zelena) i centralnim konačnim razlikama (crvena).

Na slici 5. grafički je prikazana interpretacija formula za aproksimaciju konačnim razlikama unaprijed (označena plavom bojom) i unazad (označena zelenom bojom). Nagib pravca t odgovara točnoj vrijednosti derivacije u točki C. Nagib pravca t_-, koji prolazi kroz točke A i C, odgovara formuli za aproksimaciju derivacije konačnim razlikama unazad, a nagib pravca t_+, koji prolazi kroz točke B i C, odgovara formuli za aproksimaciju derivacije konačnim razlikama unaprijed. Također, na slici možemo vidjeti da ćemo bolju aproksimaciju nagiba tangente u točki C dobiti ako promatramo nagib pravca kroz točke A i B (označen crvenom bojom na slici 5). Nagib tog pravca, odnosno aproksimaciju derivacije u točki C, možemo odrediti kao

$$f'(a) \approx \frac{f(a+h) - f(a-h)}{2h}$$

i taj izraz predstavlja formulu za **aproksimaciju derivacije centralnim (središnjim) konačnim razlikama (eng. *central difference*)**. Ova formula daje bolju aproksimaciju derivacije iz razloga što koristi tri točke (točku $(a, f(a))$ i po jednu točku sa svake njene strane)

ZADATAK 4.1.

1. Napišite funkciju za izvođenje metode konačne razlike za aproksimaciju derivacije gdje možete odabrati koju ćete metodu koristiti za određivanje derivacije (naprijed, natrag ili centralna).
2. Pronađite ugrađenu *Python* funkciju koja aproksimira derivaciju funkcije u danoj točki.
3. Primijenite metodu konačne razlike za izračun aproksimacije derivacije i usporedite sa stvarnom vrijednošću derivacije u toj točki i vrijednošću dobivenom putem *Python* ugrađene funkcije. Za primjenu koristite funkciju $f(x) = x^2 + 2x$ čija je derivacija dana funkcijom $f'(x) = 2x + 2$. Koja metoda daje najbolju aproksimaciju derivacije i putem koje metode ugrađena *Python* funkcija izračunava aproksimaciju derivacije?

RJEŠENJE ZADATKA

```
from scipy.misc import derivative

def f(x):
    return x**2 + 2*x

def h(x): #funkcija h(x) predstavlja derivaciju funkcije f(x)
    return 2*x + 2

def der (f, a, metoda ='centralna', h = 0.01):
    if metoda == 'centralna':
        return (f(a + h) - f(a - h))/(2*h)
    elif metoda == 'naprijed':
        return (f(a + h) - f(a))/h
    elif metoda == 'natrag':
        return (f(a) - f(a - h))/h
    else:
        raise ValueError("Metoda mora biti 'centralna', 'naprijed' or 'natrag'.")


print(der (f, 1)) #vrijednost derivacije dobivena korištenjem funkcije der
print(der (f, 1, dx = 0.01)) #vrijednost derivacije dobivena korištenem ugrađene funkcije u scipy.misc mod
ul
print(h (1.0))
```

ZADATAK 4.2.

1. Iz podataka za vrijednosti vremena i položaja tijela u određenom trenutku spremljenih u datoteku *ubrzano_podaci.csv* napišite program koji će odrediti derivacije položaja u ovisnosti o vremenu za svaku zadanu točku.

NAPOMENA: Da biste mogli odrediti derivacije, morate učitati podatke iz datoteke u listu i pretvoriti u dvije liste: jednu koja sadrži vrijednosti vremena (lista t) i drugu koja sadrži vrijednosti položaja (lista x).

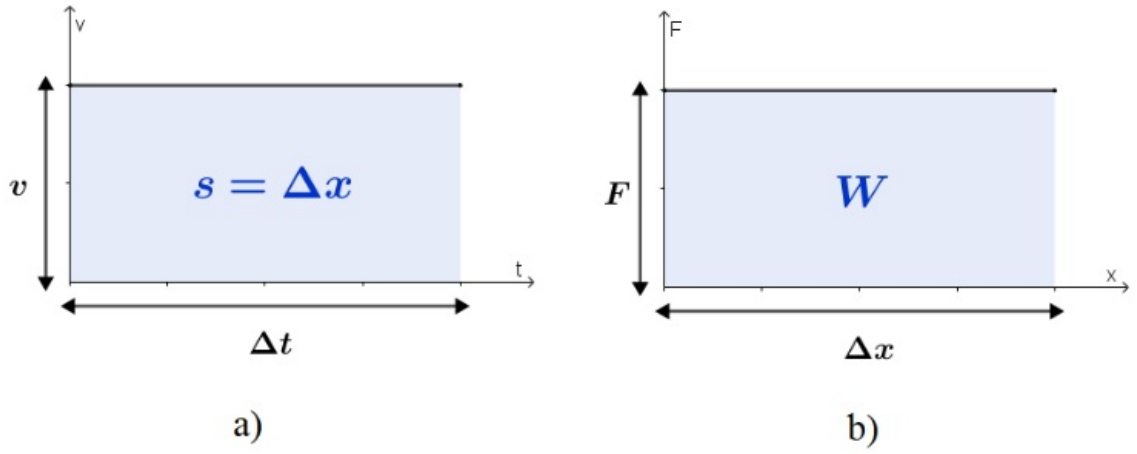
1. Nacrtajte graf ovisnosti derivacije položaja o vremenu. Koju fizikalnu veličinu predstavlja derivacija položaja tijela u vremenu?

RJEŠENJE ZADATKA

```
import csv
import matplotlib.pyplot as plt

with open('ubrzano_podaci.csv') as datoteka:
    podaci = list(csv.reader(datoteka))

for a in podaci:
    br_elementi = len (a)

t = [float(podaci[0][i]) for i in range(br_elementi)]
x = [float(podaci[1][i]) for i in range(br_elementi)]

derivacija = []
prva = (x[1]-x[0])/(t[1]-t[0])
derivacija.append(prva)


for i in range(1, len(x)-1):
    f_deriv = (x[i+1] - x[i-1])/(t[i+1] - t[i-1])
    derivacija.append(f_deriv)
    i += 1

zadnja = (x[len(x)-1]-x[len(x)-2])/(t[len(x)-1]-t[len(x)-2])
derivacija.append (zadnja)

print(derivacija)

plt.scatter(t, derivacija, s = 10)
plt.title('Derivacija položaja u vremenu')
plt.xlabel('t [s]')
plt.ylabel('dx/dt = v [m/s]')
plt.show()
```

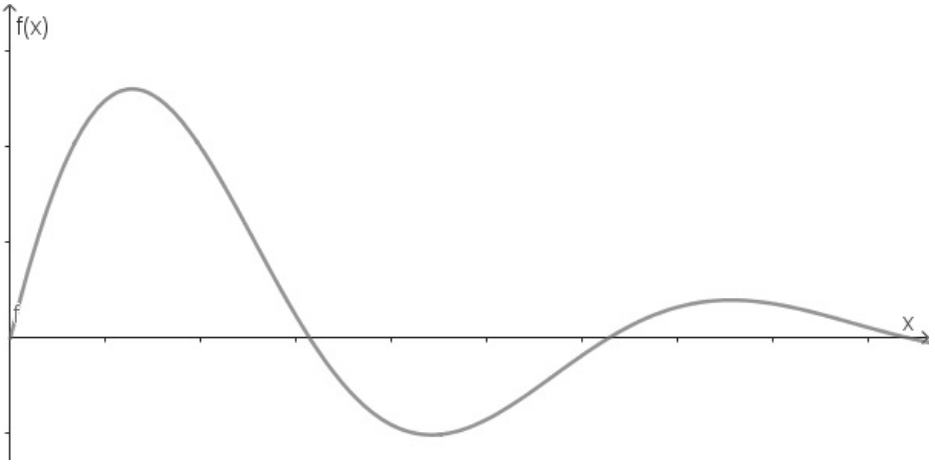
Poglavlje 5: Numerička integracija

Vrlo često se kod analize grafičkih prikaza ovisnosti neke dvije veličine može saznati nekakva dodatna informacija iz površine ispod krivulje koja predstavlja ovisnost dviju veličina. Neki od najpoznatijih primjera u fizici su određivanje površine u $v-t$ grafičkom prikazu gdje nam vrijednost površine odgovara prijeđenom putu ili pomaku tijela (Slika 1) ili u $F-x$ grafičkom prikazu gdje površina predstavlja rad W (odnosno promjenu energije tijela ΔE) (Slika 2). U slučajevima kada je ovisnost dvije veličine jednostavna (npr. pravac) ili se može podijeliti na jednostavne ovisnosti u određnim intervalima, određivanje površine ispod krivulje nije problem - račun se svodi na elementarno poznavanje matematike.



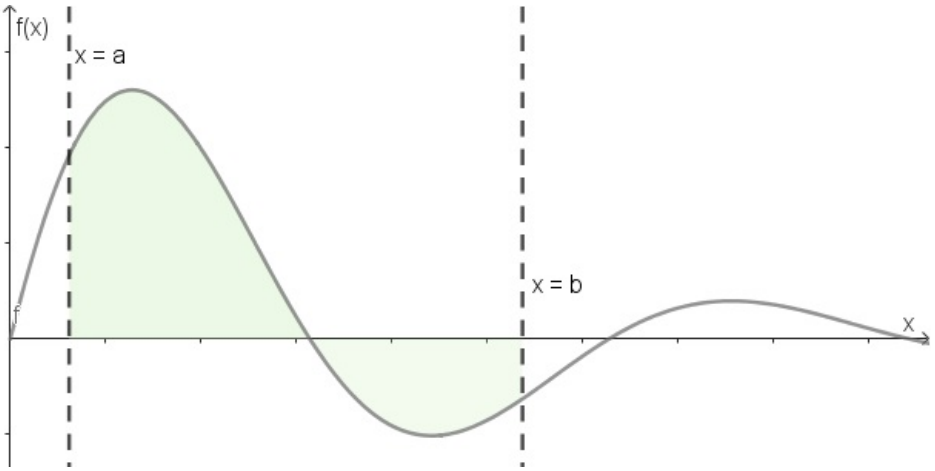
Slika 1.
a) Površina između krivulje i apscise na $v-t$ grafu predstavlja prijeđeni put/pomak tijela i b) površina između krivulje i apscise na $F-x$ grafu predstavlja rad obavljen nad tijelom.

No, što u slučaju kada područje ispod krivulje ne možemo podijeliti na jednostavne oblike kojima znamo izračunati površinu? Recimo, kao u slučaju prikazanom na Slici 2. Odgovor na ovaj problem nam daju **integrali**.

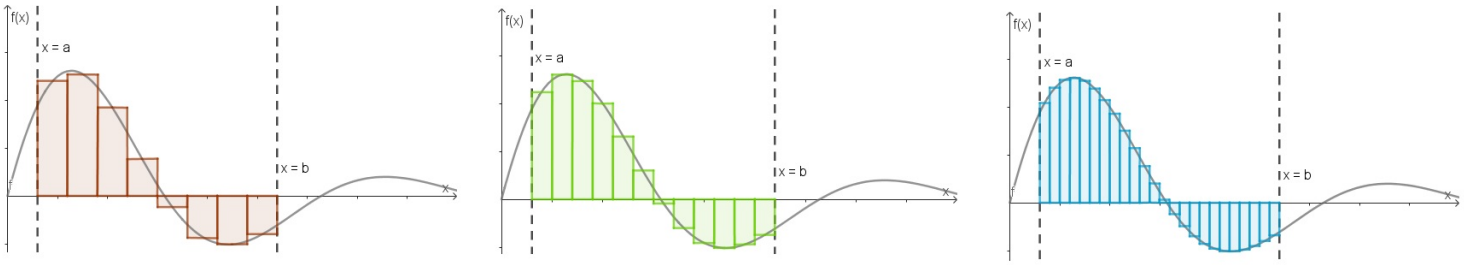


Slika 2.
Grafički prikaz funkcije kod koje ne možemo površinu odrediti elementarnim poznavanjem matematike.

Ideju integriranja su krajem 17. stoljeća oblikovali Isaac Newton i Gottfried Wilhelm Leibniz. Pojam integriranja ima dva smisla. Integriranje je, najjednostavnije rečeno, pronalaženje površine neke omeđene funkcije što postižemo računanjem tzv. **određenog integrala**. Određeni integral oblika $$\int_{a}^{b} f(x)dx \$$ predstavlja površinu omeđenu funkcijom $f(x)$, x-osi te pravcima $x=a$ i $x=b$ (granice određenog integrala). Opisana površina za funkciju $sin(x)$ je prikazana na Slici 3. Takav određeni integral si možemo zamisliti kao da traženu površinu podijelimo na pravokutnike (ili neke druge jednostavnije oblike) te ćemo zbrajanjem njihovih pojedinih površina dobiti iznos površine koju tražimo. Što je manja širina pojedinačnih pravokutnika (odsječaka) to je zbroj površina odsječaka bliži traženoj površini (prikazano na Slici 4).



Slika 3.
Grafički prikaz funkcije i površine u intervalu [a,b].



Slika 4.
Grafički prikaz približavanja traženoj površini zbrajanjem površina sve užih odsječaka.

Drugi smisao određivanje integrala je pronalaženje *antiderivacije* neke funkcije $f(x)$, odnosno njene *primitivne funkcije* oblika $F(x)$. U tom slučaju računamo **neodređeni integral** (nisu definirane granice integracije nego je dobivena općenita funkcija $F(x)$ čijom derivacijom dobivamo početnu fukciju $f(x)$)

$$F(x) = \int f(x)dx.$$

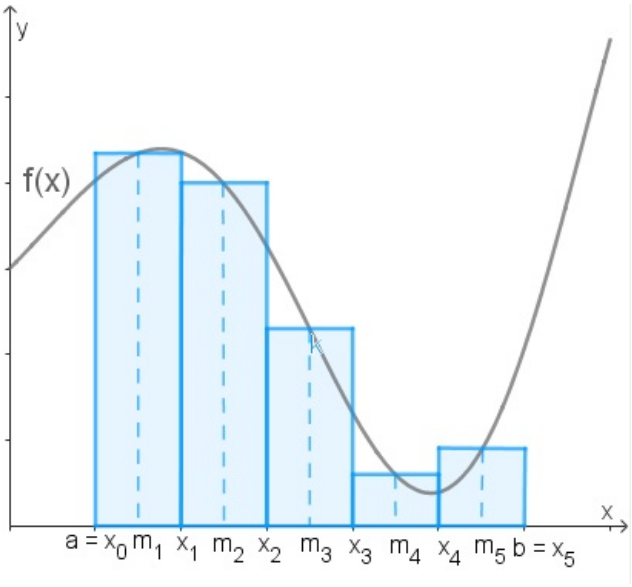
U nekim slučajavima teško je ili čak nemoguće analitički pronaći antiderivaciju funkcije pa ne možemo niti odrediti određeni integral, odnosno izračunati neku omeđenu površinu. U tim slučajevima se koristimo numeričkom integracijom takvih određenih integrala.

5.1. Metode numeričke integracije

Postoji nekoliko metoda numeričke integracije:

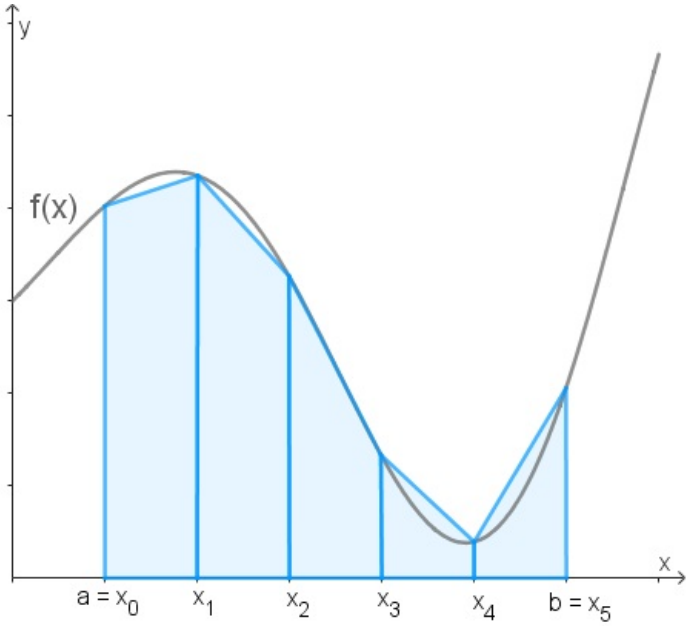
1. **pravilo srednje točke (eng. *midpoint rule*)**
2. **pravilo trapeza**
3. **Simpsonovo pravilo**

Pravilo srednje točke je najjednostavnija od navedenih metoda. Unutar te metode aproksimiramo površinu između krivulje funkcije $f(x)$ i x-osi zbrajanjem površina pravokutnuka jednakih širina čije su središnje točke ujedno i točke koje se nalaze na funkciji $f(x)$ koje određuju visinu pravokutnika (Slika 5).

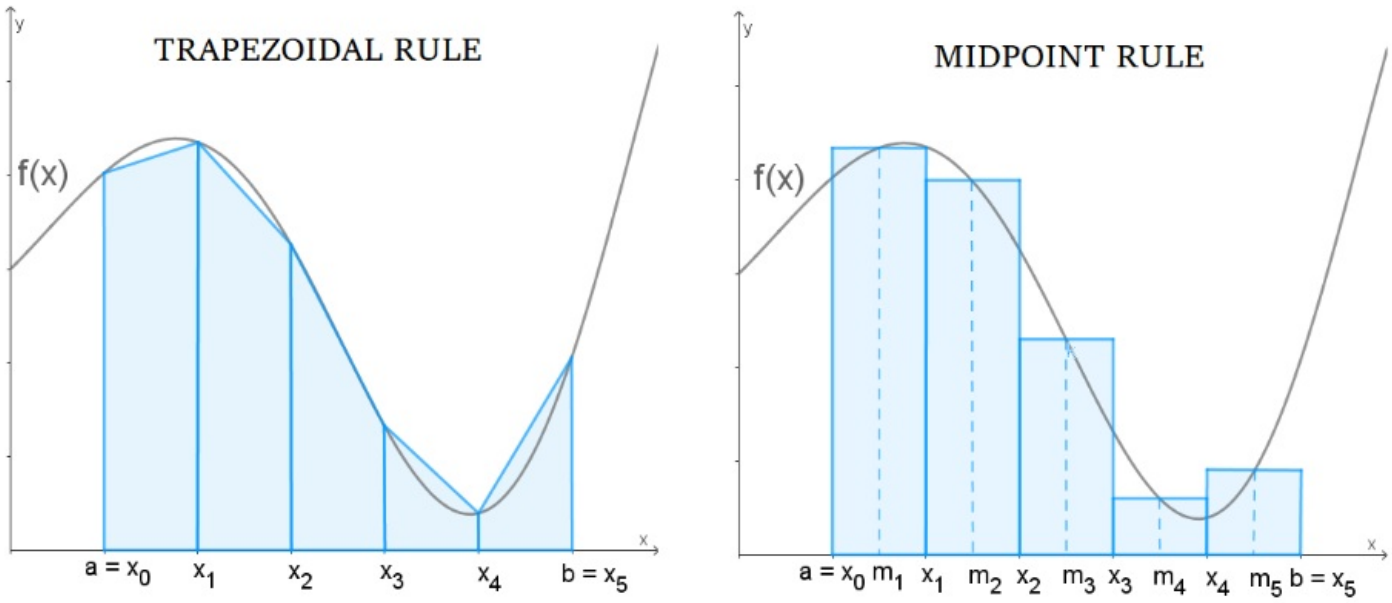


Slika 5.
Prikaz principa pravila srednje točke.

U nekim slučajevima nije moguće korisiti pravilo srednje točke, kao npr. kod aproksimacije integrala za podatke mjerenja gdje nije poznata funkcijska ovisnost dviju veličina jer u tom slučaju nije moguće odrediti "visinu pravokutnika". Umjesto pravokutnika, za aproksimaciju površine na jednakim podintervalima možemo uzeti trapeze (Slika 6) što se primjenjuje unutar metode trapeza. Ova metoda je slična metodi srednje točke, ali da bismo došli do željene preciznosti kod metode srednje točke nam je potreban velik broj pravokutnika dok aproksimacijom pomoću trapeza brže postižemo jednaku preciznost. Zanimljiv prikaz razlika u ovim dvjema metodama se nalazi na sljedećem linku (https://www.whitman.edu/mathematics/calculus_online/section08.06.html). Ako promotrimo Sliku 7 možemo vidjeti da metodom trapeza se precijenjuje vrijednost integrala na intervalima gdje je funkcija konkavna, odnosno podcijenjuje vrijednost na intervalima gdje je funkcija konveksna. S druge strane, kod metode srednje točke te pogreške se "usrednje" jer se precjenjivanje i podcjenjivanje vrijednosti intergrala događa na istim dijelovima funkcije. Upravo to je jedan od razloga zašto je općenito metoda srednje točke preciznija od metode trapeza kod aproksimacije vrijednosti određenog intergala neke funkcije.



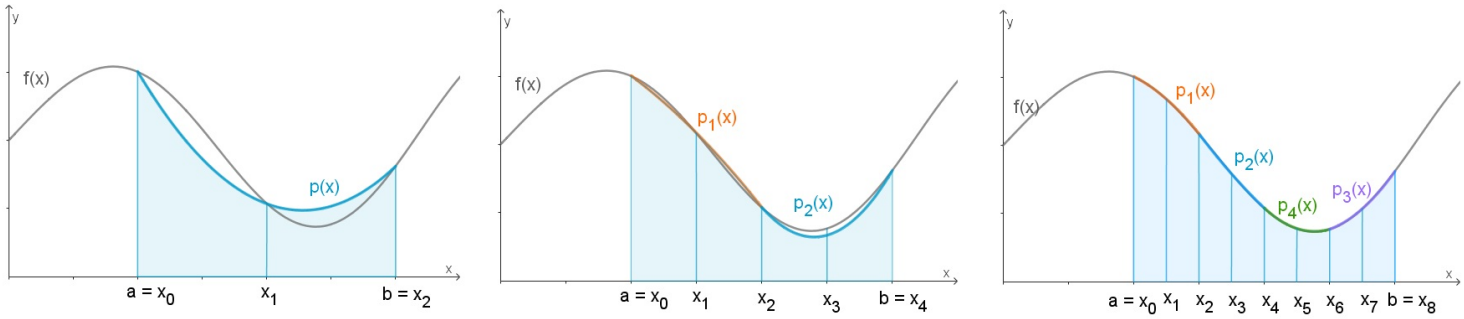
Slika 6.
Prikaz principa pravila trapeza.



Slika 7.

Usporedba metode trapeza (lijevo) i metode srednje točke (desno) koja pokazuje zašto metodom srednje točke dobivamo veću točnost nego metodom trapeza.

Kod pravila srednjih točaka i pravila trapeza, integral se po dijelovim funkcije aproksimira pomoću konstantne funkcije (pravilo srednje točke), odnosno linearne funkcije (pravilo trapeza). Simpsonovo pravilo za aproksimaciju vrijednosti integrala koristi polinom drugog stupnja, tj. kvadratnu funkciju. Interval unutar kojega želimo odrediti integral podijelimo na paran broj podintervala jednake širine. Integral funkcije $f(x)$ na prvom paru podintervala aproksimiramo integralom polinoma $p(x) = Ax^2 + Bx + C$ koji prolazi kroz točke $(x_0, f(x_0))$, $(x_1, f(x_1))$ i $(x_2, f(x_2))$. Na sljedećem paru podintervala integral aproksimiramo integralom novog polinoma p_2 koji prolazi kroz sljedeće tri točke. Postupak se ponavlja za svaki sljedeći par podintervala. Princip na kojem radi Simpsonovo pravilo prikazan je na Slici 8.



Slika 9.

Prikaz aproksimacije određenog integrala Simpsonovim pravilom.

5.2. Kako aproksimirati integrale pomoću opisanih metoda?

Svaka od ovih metoda za aproksimaciju koristi podintervale jednakih širina h na koje je podjeljeno područje integracije, interval $[a, b]$. Da bismo dobili širinu svakog podintervala moramo širinu intervala $[a, b]$ podijeliti na željeni broj intervala $(n - 1)$ gdje n predstavlja broj točaka pomoću kojih dijelimo interval. Naravno, pritom mora vrijediti uvjet da je $b > a$.

$$h = \frac{b - a}{n - 1}$$

5.2.1. Metoda srednje točke

Kod metode srednje točke širina pravokutnika je dana kao $h = \frac{b-a}{n-1}$, kao što je opisano ranije, a visina pravokutnika za svaki pointerval je određena s vrijednošću funkcije u točki koja je sredina granica pointervala (kao što prikazuje Slika 6). Ako se općenito radi o podintervalu $[x_i, x_{i+1}]$, srdnja točka tog podintervala je dana kao:

$$y_i = \frac{x_{i+1} + xi}{2}$$

. Naravno, površina pravokutnika je onda određena kao umnožak širine i visine za svaki pojedini podinterval. Zbrajanjem površina pravokutnika dobivenih konačno dobivamo aproksimaciju određenog integrala, odnosno

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} hf(y_i)$$

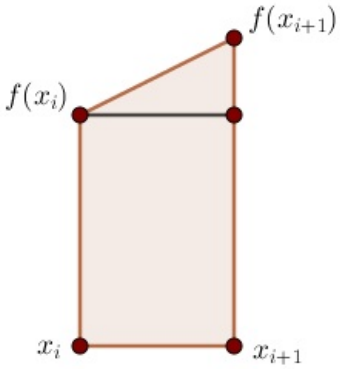
.

5.2.2. Metoda trapeza

Kao i kod metode srednje točke, širina podintervala (širina pojedinog trapeza) je h. U ovom slučaju je potrebno odrediti površinu svakog trapeza i zbrojiti sve te površine kako bismo dobili aproksimaciju integrala. Ako pogledamo jedan od trapeza na Slici 8 možemo vidjeti da njegovu površinu možemo rasčlaniti na površinu pravokutnika i pravokutnog trokuta kao što je prikazano na Slici 9. Pritom za površine pravokutnika i pravokutnog trokuta vrijedi sljedeće:

$$P_{pravokutnik} = hf(x_i),$$

$$P_{trokut} = \frac{h(f(x_{i+1}) - f(x_i))}{2}.$$



Slika 9.
Prikaz trapeza koji se može podijeliti na pravokutnik i pravokutni trokut.

Gledajući na taj način, površina svih trapeza je određena kao $P = \sum_{i=0}^{n-1}(P_{pravokutnik} + P_{trokut})$. Dobiveni izraz se može još i dalje srediti pa konačno dobivamo

$$\int_a^b f(x)\,dx \approx \frac{h}{2}[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)]$$

.

Točke x_0 i x_n zapravo predstavljaju granice određenog integrala a i b.

5.2.3. Simpsonova metoda

Posljednja metoda aproksimira određeni integral pomoću polinoma drugog stupnja (što je grafički parabola). U ovom slučaju aproksimacija se radi kroz tri točke, odnosno kroz parove podintervala. Za određivanje ove aproksimacije nam je potrebna malo kompleksnija matematika pa ćemo je samo navesti u konačnom obliku:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}[f(x_0) + 4\sum_{i=1,neparni}^{n-1} f(x_i) + 2\sum_{i=2,parni}^{n-2} f(x_i) + f(x_n)]$$

.

Kod ove metode je vrlo važno da postoji paran broj podintervala, odnosno nepran broj točaka pomoću kojih se dijeli inteval $[a, b]$. Kao i u prethodnoj metodi, vrijedi da $x_0 = a$ i $x_n = b$.

ZADATAK 5.1.

1. Definirajte funkciju koja računa integral neke funkcije $f(x)$ pomoću metode srednje točke. Neka funkcija ima ulazne parametre a, b i n, gdje a i b predstavlju donju i gornju granicu integracije, a n broj točaka pomoću kojih djelimo područje integracije na intervale.
2. Izračunajte integral funkcije $f(x) = sin(x)$ u intervalu $[0, \pi]$.
3. Usporedite dobivenu vrijednost s točnom vrijednosti integrala unutar tog intervala: $\int_0^{\pi} sin(x)\,dx = 2$.

RJEŠENJE ZADATKA

```
import numpy as np

def f(x):
    return np.sin(x)

def int_mid(a,b,n):
    x = np.linspace (a, b, n)
    h = (b-a)/(n-1) #n je broj točaka, a broj intervala je (n-1)
    sum_der = 0
    i = 0
    for i in range (n-1):
        sum_der = sum_der + h*f((x[i+1]+x[i])/2)
    print(sum_der)

int_mid(0, np.pi, 11)
```

ZADATAK 5.2.

1. Definirajte funkciju koja računa integral neke funkcije $f(x)$ pomoću metode trapeza. Neka funkcija ima ulazne parametre a, b i n, gdje a i b predstavlju donju i gornju granicu integracije, a n broj točaka pomoću kojih djelimo područje integracije na intervale.
2. Izračunajte integral funkcije $f(x) = sin(x)$ u intervalu $[0, \pi]$.
3. Izračunate vrijednost tog istog integrala preko Pythonove ugrađene funkcije *quad* unutar paketa ***scipy.integrate***. Usporedite dobivenu vrijednost s vašom vrijednosti.

RJEŠENJE ZADATKA

```
import numpy as np
import scipy.integrate as integrate

def f(x):
    return np.sin(x)

def int_trap(a,b,n):
    x = np.linspace (a, b, n)
    h = (b-a)/(n-1)
    sum_der = 0
    i = 1
    for i in range (n-1):
        sum_der = sum_der + h/2 * (f(a) + 2*f(x[i]) + f(b))
    print (sum_der)

int_trap(0, np.pi, 11)
print(integrate.quad(f,0,np.pi))
```

ZADATAK 5.3.

1. Definirajte funkciju koja računa integral neke funkcije $f(x)$ pomoću Simpsonove metode. Neka funkcija ima ulazne parametre a, b i n, gdje a i b predstavlju donju i gornju granicu integracije, a n broj točaka pomoću kojih djelimo područje integracije na intervale.
2. Izračunajte integral funkcije $f(x) = sin(x)$ u intervalu $[0, \pi]$.
3. Izračunate vrijednost tog istog integrala preko Pythonove ugrađene funkcije *quad* unutar paketa ***scipy.integrate***. Usporedite dobivenu vrijednost s vašom vrijednosti.

RJEŠENJE ZADATKA

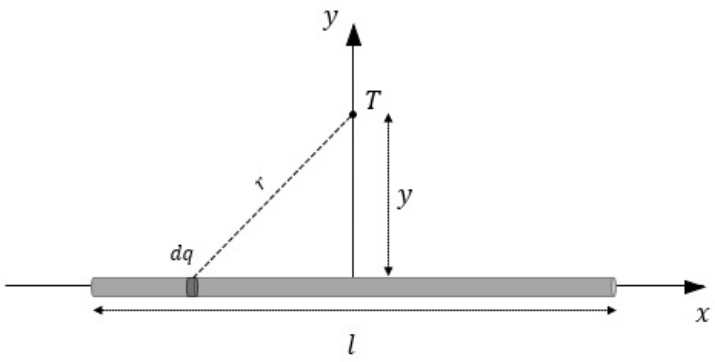
```
import numpy as np
import scipy.integrate as integrate

def f(x):
    return np.sin(x)

def int_simp (a,b,n):
    x = np.linspace (a, b, n)
    h = (b-a)/(n-1) # n mora biti neparan da bi broj intervala bio paran - uvjet za ovo pravilo!
    suma_parni = 0
    suma_neparni = 0
    i = 1
    for i in range (n-1):
        if i % 2 == 0:
            suma_parni = suma_parni + 2*f(x[i])
        else:
            suma_neparni = suma_neparni + 4*f(x[i])
        suma_svi =  h/3 *(f(x[0]) + suma_neparni + suma_parni + f(x[n-1]))
    print (suma_svi)


int_simp (0, np.pi, 100)
integrate.quad(f,0,np.pi)
```

5.3. Fizikalni problem: Određivanje potencijala linearne raspodjele naboja

Na štapu duljine l raspoređen je naboj tako da je linearna gustoća naboja (odnosno, naboj po jedinici dujine) dan kao

$$\lambda(x) = \frac{Q}{l} e^{-\frac{x^2}{l^2}}.$$

Za navedenu gustoću naboja je potrebno odrediti električni potencijal u točki T na visini y točno iznad polovišta štapa. Zbog jednostavnosti si možemo postaviti štap tako da se njegovo polovište nalazi u ishodištu koordinatnog sustava kao što prikazuje Slika 10.



Slika 10.
Prikaz postava problema u koordinatnom sustavu.

Znamo da potencijal točkastog naboja ovisi o iznosu naboja koji stvara taj potencijal q i o udaljenosti promatrane točke prostora od naboja r, odnosno

$$V(r) = \frac{1}{4\pi\varepsilon_0}\frac{q}{r}.$$

U našem slučaju nemamo samo jedan točkasti naboj nego mnogo njih raspoređeno po štapu (što je određeno linearnom gustoćom naboja $\lambda(x)$) pri čemu svaki naboj na štapu doprinosi ukupnom potencijalu. Možemo to reći i drugačije, neki mali djelić naboja dq u točki T stvara potencijal $dV = \frac{1}{4\pi\varepsilon_0}\frac{dq}{r}$. Oznake dq i dV predstavljaju *diferencijale* naboja i potencijala, odnosno njihove proizvoljno male veličine. Dakle, ukupan potencijal u točki T će biti jednak zbroju svih tih proizvoljno malih potencijala što nas navodi na intergal

$$V = \int_a^b dV = \int_a^b \frac{1}{4\pi\varepsilon_0}\frac{dq}{r}$$

gdje granice intergala a i b predstavljaju lijevi i desni kraj štapa, tj. $a = -\frac{l}{2}$ i $b = \frac{l}{2}$ jer integriramo po x koordinati. Da bismo vidjeli kako se mijenja potencijal ovisno o x koordinati možemo diferencijal naboja zapisati preko linearne gustoće naboja. S obzirom da znamo da općenito vrijedi za linearnu gustoću naboja $\lambda(x) = \frac{Q}{x}$, možemo vidjeti da je prema tome $Q = \lambda(x)x$ ili u našem sluačju gdje promatramo diferencijal naboja $dq = \lambda(x)dx$. Također znamo da potencijal ovisi i o udaljenosti promatrane točke od naboja koji ga stvara, a u našem slučaju se udaljenost mijenja kako se pomičemo po štapu. Točnije, udaljenost ovisi o x kooordinati što možemo vidjeti iz Slike 11. gdje udaljenost r možemo dobiti koristeći se Pitagorinim poučkom pri čemu vrijedi da je $r = \sqrt{x^2 + y^2}$. Kada sve to uklopimo, dobivamo integral koji je potrebno izračunati kako bismo odrediti potencijal u točki T,

$$V = \int_{-l/2}^{l/2} \frac{1}{4\pi\varepsilon_0}\frac{\lambda(x)}{\sqrt{x^2 + y^2}}\,dx$$

$$V = \int_{-l/2}^{l/2} \frac{1}{4\pi\varepsilon_0} \frac{\frac{Q}{l} e^{-\frac{x^2}{l^2}}}{\sqrt{x^2+y^2}} dx$$

$$V = \frac{1}{4\pi\varepsilon_0} \frac{Q}{l} \int_{-l/2}^{l/2} \frac{e^{-\frac{x^2}{l^2}}}{\sqrt{x^2+y^2}} dx.$$

Prikazani intergal nije jednostavno analitički izračunati, niti mi imamo dovoljno matematičkog znanja da bismo ga mogli analitički izračunati, ali ga možemo numerički izračunati pod određenim uvjetima (poznavajući vrijednosti konstanti ε_0, l, Q i znajući udaljenost y od štapa na kojoj želimo odrediti vrijednost potencijala).

ZADATAK 5.4.

Numerički odredite vrijednost električnog potencijala (ili preko Pythonove ugrađene funkcije ili preko funkcije koju ste samostalno napisali) u točki T koja se nalazi na visini u točki

$$y = 4m$$

iznad polovišta štapa za ranije prikazani fizikalni problem određivanja električnog potencijala za linearnu gustoću naboja oblika $\lambda(x) = \frac{Q}{l} e^{-\frac{x^2}{l^2}}$ poznavajući vrijednosti:

$$l = 2m$$

$$\frac{Q}{4\pi\varepsilon_0 l} = 1V.$$

RJEŠENJE ZADATKA

```
import math
import scipy.integrate as integrate

y = 4
l = 2
faktor = 1

def f(x):
    return (math.exp(-(x*x)/(l*l)))/(math.sqrt(x*x+y*y))

vrijednost, pogreska = integrate.quad(f,-l/2,l/2)

potencijal = faktor * vrijednost

print(potencijal)
print(pogreska)
```

ZADATAK 5.5. (ne baš tako jednostavan zadatak)

Numerički odredite vrijednost električnog potencijala u točki T za ranije prikazani fizikalni problem određivanja električnog potencijala za linearnu gustoću naboja oblika $\lambda(x) = \frac{Q}{l} e^{-\frac{x^2}{l^2}}$ poznavajući vrijednosti:

$$l = 2m$$

$$\frac{Q}{4\pi\varepsilon_0 l} = 1V,$$

ali u intervalu 1 m $< y < 10$ m. Dobivene vrijednosti spremite u listu i grafički prikažite ovisnost potencijala o udaljenosti y od polovišta štapa.

RJEŠENJE ZADATKA

```
import math
import scipy.integrate as integrate

l = 2
faktor = 1
y0 = 1
yn = 10
potencijal =[]
udaljenost =[]

for y in range (y0,yn+1,1):
    udaljenost.append(y)

def f(x):
    return (math.exp(-(x*x)/(l*l)))/(math.sqrt(x*x+y*y))


for y in range (y0,yn+1,1):
    vrijednost, pogreska =  integrate.quad(f,-l/2,l/2)
    pot = faktor * vrijednost
    potencijal.append(pot)

print (udaljenost)
print (potencijal)

V_y = list(zip(udaljenost,potzencijal))
list_plot(V_y, color = 'blue', title = 'Ovisnost potencijala o udaljenosti od štapa', axes_labels = ['$y/m
$','$V/V$'], marker = "s")
```

Zadatak 5.6.

Sila $F(x) = F_0 - kx$ djeluje u $+x$ smjeru na tijelo mase m koje se nalazi na horizonatlnoj podlozi bez trenja. Kutija je početno mirovala na položaju x_0. Koliki je rad obavljen nad kutijom ako je rad nad njom obavljan do trenutka kada se ona nalazila na položaju x?

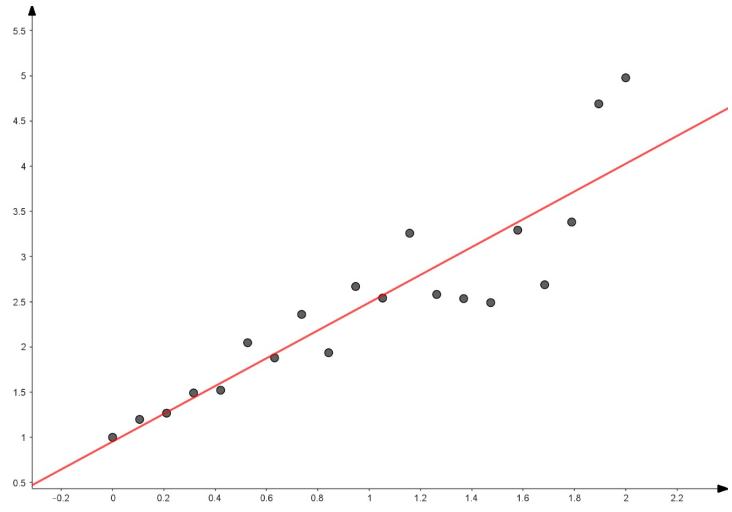
1. Razmislite kako biste odredili rad obavljen nad kutijom pomoću numeričkog integriranja (raspišite si izraz za rad u obliku diferencijala rada prema primjeru s nabojem i određivanjem potencijala) te napišite program koji računa općenito računa rad za ovaj problem.
2. Izračunajte rad za sljdeće vrijednosti konstanti: $F_0 = 18$ N, $k = 0,53$ N/m, $m = 6$ kg, $x_0 = 0$ m i $x = 14$ m.

RJEŠENJE ZADATKA

```
import numpy as np
import scipy.integrate as integrate

F0 = float(input("Za F(x)=F0-kx unesite vrijednost F0 [N]:"))
k = float(input("Za F(x)=F0-kx unesite vrijednost k [N/m]:"))


def f(x):
    return F0-k*x

def int_simp (a,b,n):
    x = np.linspace (a, b, n)
    h = (b-a)/(n-1)
    suma_parni = 0
    suma_neparni = 0
    i = 1
    for i in range (n-1):
        if i % 2 == 0:
            suma_parni = suma_parni + 2*f(x[i])
        else:
            suma_neparni = suma_neparni + 4*f(x[i])
        suma_svi =  h/3 *(f(x[0]) + suma_neparni + suma_parni + f(x[n-1]))
    print (suma_svi)


int_simp (0, 14, 10000)
integrate.quad(f,0,14)
```

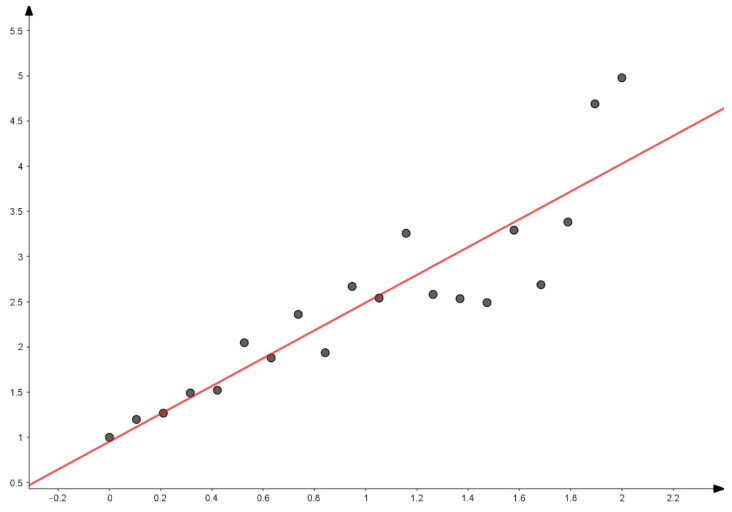
Poglavlje 7: Numeričke metode koje se koriste za linearnu regresiju i metodu najmanjih kvadrata

U znanosti se često susrećemo s velikom količinom podataka koju je potrebno analizirati. Osnova analize nekakvog seta podataka je traženje ovisnosti između promatranih varijabli - ovisi li jedna varijabla o drugoj i kakva je njihova ovisnost. Ta ovisnost nam je često nepoznata i teško vidljiva direktno iz eksperimentalnih podataka zbog njihovog raspšenja kao posljedice postojanja slučajnih pogrešaka. Zato je potrebno statističkom obradom podataka odabrati ovisnost koja najbolje odgovara danom setu podataka mjerenja. Određivanje funkcionalne ovisnosti između dviju ili više varijabli ispituje regresijska analiza. Regresijskom analizom na temelju utvrđene povezanosti i poznate vrijednosti nezavisne varijable predviđamo vrijednost zavisne varijable - točnije, daje nam vezu između zavisne i nezavisne varijable. Takva analiza se često koristi upravo za predviđanja ponašanja nekih pojava, ali i za zaključivanje uzročnih veza između zavisne i nezavisne varijable. Ovisno o broju varijabli čiju ovisnost provjeravamo, regresijsku analizu možemo podijeliti na jednostavnu i višestruku. Također je možemo podijeliti i na linearnu te nelinearnu regresijsku analizu (slike 1 i 2 prikazuju linearnu i nelinearnu povezanost dviju varijabli). .



Slika 1.

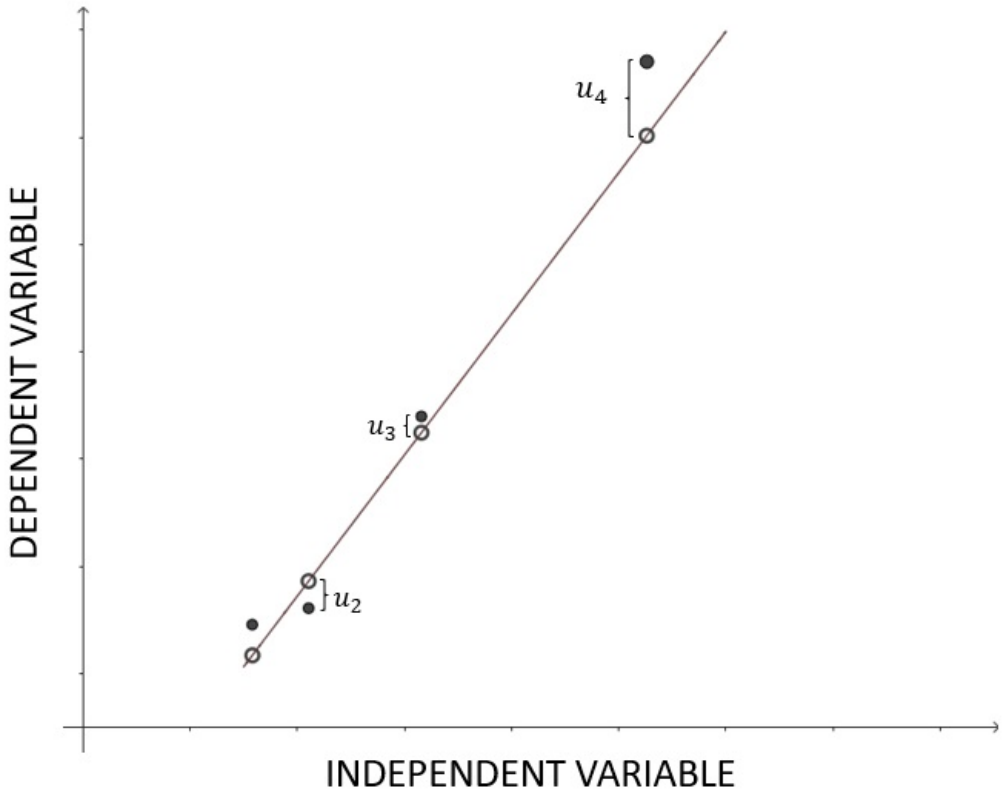
Linearna povezanost dvije varijable.



Slika 2.

Neinearna povezanost dvije varijable.

Najpoznatije metode regresijske analize su *linearna regresija* i *metoda najmanjih kvadrata*. Najjednostavnija metoda je svakako jednostavna linearna regresija kod koje se promatra utjecaj jedne nezavisne varijable (x) na jednu zavisnu varijablu (y). U tom slučaju tražimo pravac oblika $y = a_0 + a_1 x + u$ koji najbolje odgovara danom setu eskperimentalnih podataka, odnosno tražimo najvjerojatnije vrijednosti koeficijenata pravca a_0 i a_1 takve da taj pravac s najmanjim odstupanjem prolazi kroz eksperimentalne podatke. Veličina u predstavlja slučajnu pogrešku koja pokazuje razliku između eksperimentalno izmjerenih vrijednosti nezavisne varijable i teorijskog predviđanja (slika 3), takozvani *rezidual*. Kod metode najmanjih kvadrata (MNK) funkcija koja najbolje odgovara eksperimentalnim podacima se određuje tako da je zbroj kvadrata razlika između izmjerenih i izračunanih vrijednosti minimalan. Odnosno, određuje se funkcija kojoj krivulja prilazi što bliže danim točkama. Ona se može primjeniti i na linearnim i na nelinarnim ovisnostima dviju varijabli.



Slika 3.

Prikaz jednostavne linearne regresije.

Kao i kod linearne regresije, i kod metode najmanjih kvadrata tražimo najvjerojatnije vrijednosti koeficijenata funkcije takve da dobivena krivulja prilazi što bliže ekspemirentalno određenim vrijednostima. Najjednostavniji oblik funkcije za koji se primjenjuje metoda najmanjih kvadarata je linearna funkcija oblika $y = a_0 + a_1 x$ te se u tom slučaju mogu dobiti analitička rješenja za koeficijente a_0 i a_1, kao i za njihove pogreške M_{a_0} i M_{a_1}. Koeficijente i njihove pogreške dobivamo pomoću sljedećih izraza:

$$a_0 = \frac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$a_1 = \frac{n \cdot \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$M_{a_0} = \sqrt{\frac{1}{n-2}\left[\frac{n \cdot \sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} - a_1^2\right]}$$

$$M_{a_1} = M_{a_0} \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$$

ZADATAK 7.1.

Za skup podataka koji je definiran na početku koda prikazanog u nastavku ovog zadatka, napišite fuknkciju *procjena* čije će ulazne varijable biti *x i y*, koja pomoću ranije navedenih izraza računa koeficijente a_0 i a_1 te njihove pogreške M_{a_0} i M_{a_1}. Grafički prikažite vrijednosti *x* i *y* (kao točke) te na istom grafu prikažite i pravac čiji su koeficijenti dobiveni metodom najmanjih kvadrata. Ispravno označite osi i postavite legendu.

```
import numpy as np

x = np.linspace(0, 2, 20)
y = x + x * np.random.random(len(x))
```

RJEŠENJE ZADATKA

```
import math

def procjena(x,y):
    n = len(x)
    sum_x=np.sum(x)
    sum_y=np.sum(y)
    sum_x2=np.sum(x*x)
    sum_xy=np.sum(x*y)
    sum_y2=np.sum(y*y)

    a0=(sum_x2 * sum_y - sum_x * sum_xy)/(n*sum_x2-sum_x * sum_x)
    a1=(n*sum_xy - sum_x * sum_y)/(n*sum_x2-sum_x * sum_x)
    Ma1=math.sqrt(1/(n-2)*((n*sum_y2-(sum_y)^2)/(n*sum_x2-(sum_x)^2) - a1^2))
    Ma0=Ma1*math.sqrt(1/n * sum_x2)

    return a0,a1,Ma0,Ma1

import matplotlib.pyplot as plt

a0 = procjena(x,y)[0]
a1 = procjena(x,y)[1]

plt.figure(figsize = (10,8))
plt.plot(x, y, 'b.', label='Podaci')
plt.plot(x, a0 + a1*x, 'r', label='Prilagodba metodom najmanjih kvadrata')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc="upper left")
plt.show()
```

Već su i ovi izrazi dosta kompliciranji, a još više se zakompliciraju u slučaju polinoma trećeg stupnja (gdje je najveća potencija varijable 3). Za neke više potencije polinoma, kao i za mnoge funkcije, nije moguće pronaći analitičko rješenje za određivanje koeficijenata i njihovih pogrešaka. U tom slučaju se oslanjamo na numerička rješenja i numeričke metode određivanja koeficijenata. Tu nam pomaže Pythonova ugrađena funkcija `optimize.curve_fit()` koja je ugrađena u knjižnicu `scipy`. Ona koristi metodu najmanjih kavadrata za određivanje koeficijenata bilo kakve fuknicje (linearne i nelinearne).

ZADATAK 7.2.

Potražite na internetu dokumentaciju za funkciju `optimize.curve_fit()` i proučite kako se koristi. Primjenite je na prethodnom problemu s istim definiranim točkama te grafički prikažite i ovaj slučaj.

RJEŠENJE ZADATKA

```
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt

x = np.linspace(0, 2, 20)
y = 1 + x + x * np.random.random(len(x))

def f(x, a1, a0):
    y = a0 + a1*x
    return y

koef, pogr = optimize.curve_fit(f, xdata = x, ydata = y)


plt.figure(figsize = (10,8))
plt.plot(x, y, 'b.', label='Podaci')
plt.plot(x, koef[0]*x + koef[1], 'r', label='Prilagodba metodom najmanjih kvadrata')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc="upper left")
plt.show()
```

ZADATAK 7.3.

Učitajte datoteku *metoda1.csv* koja se nalazi u prilogu ove bilježnice i podatke iz datoteke spremite u liste. Prikažite podatke grafički i procjenite o kakvoj ovisnosti se radi te pomoću metode najmanjih kvadrata odredite koeficijente za pretpostavljeni oblik funkcije.Na kraju prikažite i funkciju dobivenu metodom najmanjih kvadrata na istom grafu kao i podatke. Dodajte nazive na osima i legendu.

RJEŠENJE ZADATKA

```
import csv

with open('metoda1.csv') as f:
    m = list (csv.reader(f))
    for a in m:
        x = [float(m[0][i]) for i in range (len(a))]
        y = [float(m[1][i]) for i in range (len(a))]
    f.close()

def func(x, a, b, c):
    y = a*np.exp(b*x) + c
    return y

koef, pogr = optimize.curve_fit(func, xdata = x, ydata = y)


plt.figure(figsize = (10,8))
plt.plot(x, y, 'b.', label = 'Podaci')
plt.plot(x, koef[0]*np.exp(koef[1]*np.array(x)) + koef[2], 'r', label='Prilagodba metodom najmanjih kvadra
ta')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc="upper left")
plt.show()
```