1. Zelo kratek uvod v programiranje

Za numerično reševanje fizikalnih problemov lahko uporabimo praktično katerega koli od programskih jezikov, ki so na voljo. Najpogosteje uporabljamo jezike kot so `C/C++`, `Matlab`, `Mathematica`, `CERN ROOT` ter `Python`, ki so podprti z bogatimi matematičnimi knjižnjicami za izvajanje najrazličnejših matematičnih operacij ter knjižnjicami za histogramiranje in risanje grafov. Te močno olajšajo našo pot od začetnih fizikalnih izrazov do končnih rezultatov.

Tu bomo uporabljali popularen programski jezik `Python` skupaj s knjižnicami NumPy, SciPy in Matplotlib. `Python` je interpreterski programski jezik, ki deluje na vseh sodobnih platformah (Windows, Linux, MacOS). K popularnosti tega programskega jezika je pripomogla zelo obširna prostodostopna dokumentacija z zledi in enostavna sintaksa, ki je podobna angleskemu jeziku.

Programiranja za potrebe modeliranja fizikalnih procesov, analize meritev, statistične obdelave velikih količin podatkov in strojnega učenja se najenostavneje lotimo z uporabo okolja `Anaconda`, ki močno poenostavi instalacijo programskega jezika, potrebnih knjižnjic in delovnega okolga na izbrani operacijski sistem.

Ko začnemo s programiranjem v `Pythonu`, je pomembno vedeti, da se vsak zapisani ukaz zaključi s prehodom v novo vrstico. To je drugače kot v programskih jezikih kot sta npr. `C/C++` in `Java`, kjer moramo vsako vrstico programa zaključiti s podpičjem. Pomembno si je tudi zapomniti, da v `Pythonu` vsebino programskih blokov oziroma struktur, kot so pogojni stavki, zanke in funkcije, določamo z zamikanjem vrstic in ne z zavitimi oklepaji, kot je to navada v drugih programskih jezikih. Komentarje programske kode v `Pythonu` označujemo z #.

Programiranja se navadno najraje lotimo s pomočjo zgledov. Najprej poskusimo na ukazno vrstico izpisati preprosta besedila. To v `Pythonu` naredimo z uporabo funkcije `print`. Kot argument funkcije navedemo niz znakov (z uporabo dvojnih narekovajev) ali število, ki ga želimo izpisati na ukazno vrstico. Funkcija nam dovoli, da izpišemo več argumentov hkrati. Kombiniramo lahko različne podatkovne tipe, npr. nize znakov in števila, ki jih moramo med seboj ločevati z vejicami.

```
1 # izpis preprostega niza znakov
```

```
print("Samo brez panike!")
# kombiniran izpis niza znakov in stevil
print("Vrednost je: ",1,". Druga vrednost je: ",2.7)
```

1.1 Matematični knjižnjici

Python ima vgrajen nabor osnovnih matematičnih funkcij, kot so `abs()`, `min()` in `max()`, a za reševanje matematičnih problemov moramo običajno vključiti matematično knjižnjico `math`. Ta vsebuje funkcije kot sta cos() in sin(), del knjižnjice pa so tudi matematične konstane kot sta π in e. Matematično knjižnjico in (tudi vse ostale) v program dodamo z ukazom `import`.

```
# vkljucimo matematicno knjiznico
import math
# Izpisemo vrednost fizikalne konstante pi:
print("Vrednost konstante pi je: ", math.pi);
# Izracunamo visino enakostranicnega trikotnika s stranico 1:
print("Visina trikotnika je: ", 1*sin(60.*math.pi/180.));
```

Še več matematičnih funkcij in zelo bogato zbirko matematičnih in fizikalnih konstant pa najdemo v knjižnjici `SciPy`. V njej najdemo tudi različne algoritme za interpolacijo, integracijo, reševanje diferencialnih enačb in problemov lastnih vrednosti ter Fourierovo transformacijo.

Za zgled demonstrirajmo uporabo fizikalnih konstant iz knjižnjice `SciPy`. Te naložimo v program s klicem `from scipy import constants`. Algoritem primerja vrednost svetlobe hitrosti v vakuumu, ki jo izračuna iz zveze $c_0 = 1/\sqrt{\varepsilon_0 \mu_0}$ in tisto shranjeno v programski knjižnjici.

```
from scipy import constants

c = constants.speed_of_light
eps0 = constants.epsilon_0
mu0 = constants.mu_0
print(c**2 * mu0 * eps0)
```

Na analogen način lahko uporabimo tudi sposebne matematične funkcije, ki jih v program pokličemo z ukazom `from scipy import special`. V spodnjem zgledu uporabimo funkcijo Erf(x) oziroma Gaussov integral in izračunamo njeno vrednost pri $x = 2$, pri tem pa vemo, da velja Erf$(2) \approx 1$.

```
from scipy import special

print('Erf(2) = ',special.erf(2))
```

1.2 Zanke

V Pythonu lahko za ciklično izvajanje programske kode uporabimo zanki `while` in `for`. Vsebina zanke `while` se izvaja toliko časa, dokler je pogoj zanke izpolnjen. Stanje pogoja se preverja na začetku vsake iteracije zanke:

```
i=1
while i < 10:
    # vsebina zanke, ki se ciklicno izvaja
    print(i)
    i = i + 1
```

Zanko `for` uporabljamo tedaj, ko želimo del programske kode iteriativno izvesti za vse elemente niza, ki ga navedemo v glavi zanke:

```
# prvi zgled:
niz=[1,2,3]
for i in niz:
    print("i-ti element niza je: ",i);
# drugi zgled:
for i in 1,2,3,4,5,6,7,8:
    print("Dvakratnik stevila ",i, " je ", 2.0*i);
# tretji zgled:
for x in range(2, 12, 2):
    print(x)
```

1.3 Funkcije

Funkcija je del programske kode, ki se izvrši, ko funkcijo pokličemo. Poleg že pripravljenih funkcij, ki jih najdemo v različnih knjižnjicah, velikokrat napišemo tudi lastne. V Pythonu funkcijo definiramo z ukazom `def`. Funkciji lahko podamo vhodne podatke oziroma argumente, iz katerih nato ona pripravi (izračuna) rezultat. Funkciji lahko podamo poljubno število vhodnih argumentov različnih tipov, ki jih ločimo z vejicami. Podatke, ki želimo, da jih funkcija vrne kot rezultat, označimo z ukazom `return`. Tudi te ločimo z vejicami.

```
# prvi zgled: funkcija, ki izracuna kvadrat argumenta
def kvadrat(x):
    return x**2

print("2 na kvadrat: ",kvadrat(2.))

# drugi zgled: funkcija ki preveri, ali je stevilo
# sodo ali liho
def sodalilih(x):
    if (x%2==0):
        return "sod"
    else:
        return "lih"
x=2
print(x, " je ", sodalilih(x))
x=3
print(x, " je ", sodalilih(x))
```

1.4 Računanje z nizi

V Pythonu za delo z nizi uporabljamo zelo zmogljivo knjižnjico `NumPy`, ki omogoča najrazličnejše operacije z matrikami in nizi, ter je opremljena z obširno zbirko matematičnih funkcij, ki omogoča, da matematične operacije hitro in učinkovito izvajamo nesporedno na nizih, brez uporabe zank.

Poglejmo si dva zgleda. V prvem izračunamo vrednost funkcije sin za niz kotov $[0, \pi/2, 3\pi/2, \pi]$. Z uporabo funkcije iz knjižnjice `NumPy`, lahko vrednosti funkcije sinus za vse kote izračunamo v eni vrstici, saj ji lahko kot argument lahko podamo cel niz kotov, funkcija pa nato vrne niz vrednosti funkcije. Z navadno funkcijo sinus iz knjižnjice `math`, dani program ne bi deloval, saj sinusna funkcija v tej knjižnjici kot argument sprejme le realna števila. Za delo s to funkcijo bi npr. potrebovali zanko `for`:

```
import numpy as np
import math
x=np.array([0, math.pi/2., 3/2*math.pi, math.pi])
y=np.sin(x)
print(y)
```

V drugem zgledu izračunajmo prvih deset elementov Fibonaccijevega zaporedja: 0, 1, 1, 2, 3, 5, 8, 13, 21, 35. V ta namen najprej generiramo prazen niz dolžine 10. Nato z uporabo zanke izračunamo vsak naslednji element zaporedja in ga vstavimo v niz, ki ga na koncu izpišemo. Pri tem upoštevamo, da ima v Pythonu prvi element niza indeks 0.

```
import numpy as np
N=10
x=np.zeros(N)
x[1]=1
i=2
while i<10:
    x[i] = x[i-1] + x[i-2]
    i=i+1
print(x)
```

1.5 Branje in pisanje datotek

Branje in pisanje podatkovnih datotek datotek je pomemben del fizikalne analize podatkov. Python ima vrsto različnih funkcij za kreiranje, branje, pisanje in brisanje datotek. Lahko uporabljamo osnovne funkcije jezika kot so `open()`, `read()` in `write()`, a glede na to, da imamo najpogosteje opraviti z urejenimi nizi numeričnih podatkov, ki so urejeni po stoplcih oziroma vrsticah, je bolj prikladno uporabiti funkcije knjižnjice `NumPy`, ki podatke v datotekah direktno pretvori v nize podatkov in obratno.

Za shranjevanje nizov podatkov v datoteke uporabimo funkcijo `savetxt`, ki ji kot argument podamo ime datoteke, niz oziroma matriko, ki jo želimo shraniti ter obliko (format) v kateri želimo podatke shraniti. Za zgled za niz celih števil x med -10 in 10 izračunajmo $y(x) = x^2 + 1$ ter nato oba niza v obliki dveh povezanih stolpcev shranimo v datoteko. Za združevanje dveh nizov v matriko z dvema stolpcema uporabimo funkcijo `column_stack`. Pri shranjevanju v datoteko `foo.csv` določimo še

obliko zapisa z ukazom `fmt='%10.5f %10.5f'`. Ta določa, da bomo vsako realno število skranili na 10 točnih mest in 5 decimalnih mest natančno, stolpca pa bosta med seboj ločena z dvema presledkoma.

```
import numpy as np
# generiramo niz stevil x med -10 in 10
x = np.linspace(-10,10,21)
# izracunamo pripadajoca stevila y
y = x**2+1
# niza x in y zdruzimo v en niz
seznam = np.column_stack([x,y])
#--- seznam (x,y) shranimo v datoteko foo.csv
np.savetxt('foo.csv', seznam, fmt='%10.5f  %10.5f')
```

Z branje datotek uporabimo funkcijo `loadtxt`, ki ji kot argument podamo ime podatkovne datoteke. V naslednjem zgledu preberemo podatke iz datoteke `foo.csv` in jih naložimo v matriko `podatki`. Vsak shranjen podatek nato z uporabo zanke `for` po vrsticah izpišemo na zaslon:

```
import numpy as np
import math
# preberemo podatke iz datoteke in jih nalozimo v niz podatki.
podatki = np.loadtxt('foo.csv')
# izpisemo vsako vrstico niza posebej
for i in range(0, len(podatki)):
    print(i, podatki[i,0], podatki[i,1])
```

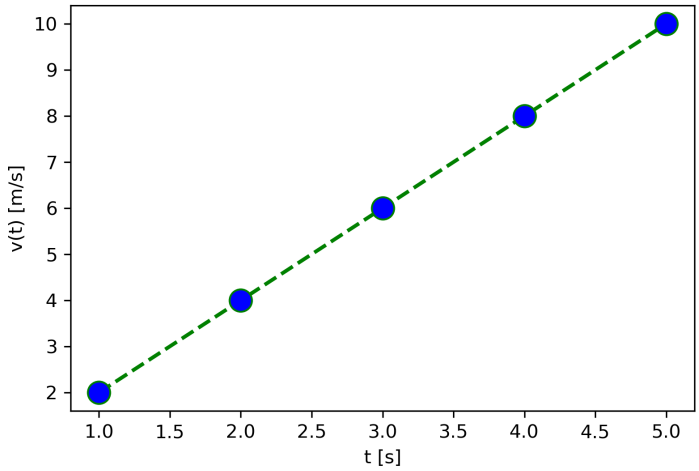
1.6 Vizualizacija podatkov

Za prikaz merskih podatkov ali rezultatov numerične analize v `Pythonu` uporabljamo knjižnjico `matplotlib`, s katero je mogoče ustvariti statične, animirane in celo interaktivne grafike. Najpomembnejši del knjižnjice je podmodul `pyplot`, ki vsebuje funkcije za prikaz podatkov. Tega navadno naložimo z ukazom `import matplotlib.pyplot as plt`, kar omogoča, da se nato v programu nanj sklicujemo z krajšim imenom ali aliasom `plt`.

Funkcija `plot()` na graf nariše točke $y(x)$. Kot argument ji podamo niza vrednosti neodvisne in odvisne spremenljivke, kot prikazuje spodnji zgled:

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

# podatki
xpoints = np.array([1, 2, 3, 4, 5])
ypoints = np.array([2, 4, 6, 8, 10])

# risanje grafa
plt.plot(xpoints, ypoints, linestyle='dashed', linewidth=2,
   color='green', marker='o', markersize=12, markerfacecolor='
   blue')
```

```
plt.xlabel("t [s]")
plt.ylabel("v(t) [m/s]")
plt.show()
```

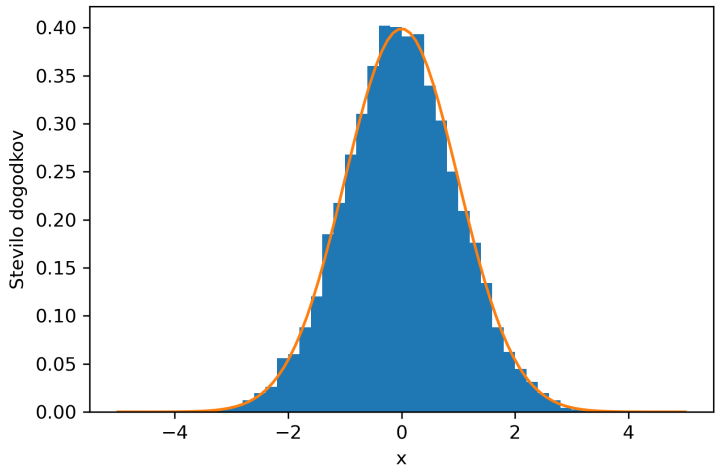


V osnovnem načinu `plot()` točke poveže s črtami. Način prikaza spremenimo z dodatnimi neobveznimi argumenti, s katerimi lahko določimo tip, debelino in barvo črte ter tip, velikost in barvo točk na grafu. Oznake oziroma opise osi nastavimo s funkcijami `xlabel` in `ylabel()`. S funkcijo `show()` graf prikažemo na zaslonu.

Za prikaz histogramov uporabljamo funkcijo `hist` podmodula `pyplot`. Funciji kot argumente podamo podatke, ki jih želimo histogramirati, število binov ter interval, na katerem želimo opazovati porazdelitev naključne spremenljivke. Če želimo porazdelitev normirati in prikazati verjetnostno gostoto, to storimo z dodatnim ukazom `density=True`. Spodnji zgled z uporabo najključnega generatorja števil iz knjižnjice `numpy` generira $N = 10000$ naključnih števil, ki so porazdeljena po normalni porazdelitvi s centralno lego pri 0 in standardno deviacijo 1. Niz števil nato hirogramira v 50 binov na intervalu $[-5,5]$ in rezultat prikažne na grafu. Za primerjavo prikaže tudi analitično izračunano normalno porazdelitev, ki jo izračunamo s funkcijo `gaussian(x, mu, sig)`:

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

N=10000

def gaussian(x, mu, sig):
    return 1/np.sqrt(2.*math.pi)/sig*np.exp(-np.power(x - mu,
   2.) / (2 * np.power(sig, 2.)))

x = np.random.normal(0, 1, N)

plt.hist(x, 50, [-5,5], density=True)
```

```
plt.xlabel("x")
plt.ylabel("Stevilo dogodkov")

t = np.linspace(-5,5,100)
a = gaussian(t, 0, 1)
plt.plot(t,a)
plt.show()
```



Za prikaz funkcij dveh spremenljivk, $f(x,y)$, lahko uporabite funkciji `contourf` in `pcolormesh`. Način uporabe funkcij demonstrira spodnji zgled. Najprej izberemo intervala spremenljivk x in y, nad katerima želimo funkcijo izračunati. Nato iz danih 1D intervalov z ukazom `meshgrid` knjižnjice `NumPy`, naredimo dvodimenzionalno matriko. Element matrike na mestu (i,j) določa vrednost koordinate x oziroma y v izbrani točki. S pomočjo generiranih matrik nato za vsako točko mreže izračunamo vrednost funkcije:

$$f(x,y) = \cos(\sqrt{x^2+y^2})e^{-\frac{\sqrt{x^2+y^2}}{10}}$$

Pri računanju si za hitrejše računanje z elementi matrik pomagamo z metodami knjižnjice `NumPy`. Dobljeni rezultat je prikazan na spodnji sliki.

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

X = np.arange(-25, 25, 0.2)
Y = np.arange(-25, 25, 0.2)
X, Y = np.meshgrid(X, Y)
R=np.sqrt(np.power(X, 2.)+np.power(Y, 2.))
Z = np.cos(R)*np.exp(-R/10)

```

```
plt.figure(figsize=(6,5), dpi=500)
graf=plt.pcolormesh(X,Y,Z,cmap=cm.inferno, shading='auto')
cbar=plt.colorbar(graf, shrink=1.0, aspect=20,orientation="
    vertical")
plt.xlabel("x [m]")
plt.ylabel("y [m]")
cbar.set_label('Vrednost funkcije', rotation=270, labelpad=10)
plt.show()
```