Cvičenie 3

Rovnice z prednášky

Všeobecný drift	$\vec{v} = \frac{\vec{F}\times\vec{B}}{qB^2}$	Gradientný drift	$\vec{v}_G = -\frac{\lvert\vec{m}\rvert}{q}\frac{\vec{\nabla}B\times\vec{B}}{B^2}$
Elektrický drift	$\vec{v}_E = \frac{\vec{E}\times\vec{B}}{B^2}$	Curvature drift	$\vec{v}_C = -\frac{mv_\parallel^2}{qB^4}[(\vec{B}\cdot\vec{\nabla})\vec{B}]\times\vec{B}$

Orbitálna teória prvého rádu

$\vec{F} = q\vec{v}^{(0)}\times[\vec{r}^{(0)}\cdot(\vec{\nabla}\vec{B})]$	$\langle\vec{F}\rangle = (\vec{m}\cdot\vec{\nabla}) + \vec{m}\times(\vec{\nabla}\times\vec{B})$
$\vec{F}_\parallel = 2 \mid \vec{m} \mid \frac{\partial B_r}{\partial r}\hat{\mathbf{z}}$	$\langle\vec{F}_\parallel\rangle = -\frac{\lvert\vec{m}\rvert}{B}[(\vec{B}\cdot\vec{\nabla})\vec{B}]_\parallel$
$\vec{F}_\perp = -2 \mid \vec{m} \mid \frac{\partial B_z}{\partial r}\hat{\mathbf{r}}$	$\langle\vec{F}_\perp\rangle = - \mid \vec{m} \mid (\vec{\nabla}B)_\perp$

Úloha 1

Bittencourt, úloha 3.6, strana 91: Uvažujme toroidálne magnetické pole:

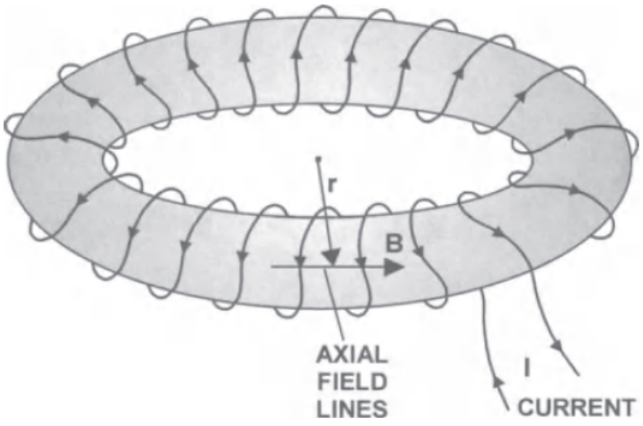

Figure 1: Magnetické pole v toroidálnej geometrii.

Úlohy:

Cvičenie 3

(i) Ukážte, že hustota magnetického toku pozdĺž tórusu je daná vzťahom $\vec{B} = B_a \left(\frac{a}{r}\right) \hat{\phi}$, kde B_a je veľkosť vektora $\vec{B}$ v radiálnej vzdialenosti $r = a$

(ii) V akom smere účinkuje gradientný drift vyvolaný radiálnou variácou B_ϕ ?

(iii) Ak $\vec{E}$ je indukované elektrické pole, v akom smere pôsobí elektrický drift $\vec{E} \times \vec{B}$?

(iv) Ukážte, že v dôsledku elektrického driftu $\vec{E} \times \vec{B}$ nie je možné udržať plazmu v čisto toroidálnej geometrii

Úloha 2

Bittencourt, úloha 3.9, strana 93: Uvažujte nekonečne dlhý vodič, ktorým tečie elektrický prúd I a je rovnomerne nabitý elektrostatickým potenciálom ϕ. Opíšte pohyb elektrónu v poli takéhoto vodiča s použitím orbitálnej teórie prvého rádu. Znázornite dráhu elektrónu a smery elektrického, gradientného a curvature driftu.

Úloha 3

V tejto úlohe si napíšeme program, ktorý numericky určuje a vykresľuje trajektóriu Zeme okolo Slnka. Program je vlastne gravitačným simulátorom. Existuje mnoho voľne dostupných gravitačných integrátorov (vyzdvihnem *ReboundX*), takže by sa zdalo, že napísať podobný program je zbytočné. Cieľom úlohy je ukázať, ako vytvoriť jednoduchý algoritmus na numerické riešenie **ľubovoľnej pohybovej rovnice**. T.j. do programu, ktorý si napíšeme bude možné vložiť, vyriešiť a vykresliť akúkoľvek pohybovú rovnicu (nie len gravitačnú), čo sa vám môže zísť v budúcnosti.

Riešenie Program, ktorý si napíšeme, je jednoduchým algoritmom simulátorov pohybových rovníc. Budeme ho formulovať v jazyku *Python*, algoritmus je však rovnaký aj pre iné programovacie jazyky. Na matematické operácie a prácu s poliami budeme využívať knižnicu *numpy*:

```
1 import numpy as np
```

Ďalším krokom je sformulovanie pohybovej rovnice. V prípade pohybu Zeme okolo Slnka máme pohybovú rovnicu Zeme v tvare

$$\vec{a}_{Earth} = -\frac{G m_{Sun} \vec{r}}{r^3}, \tag{1}$$

Cvičenie 3

kde $G = 6.674 \times 10^{-11} \frac{m^3}{kgs^2}$ je gravitačná konštanta, $m_{Sun} = 1.989 \times 10^{30} kg$ je hmotnosť Slnka, $\vec{r}$ je polohový vektor Zeme vzhľadom ku Slnku v čase t a r^3 je veľkosť vektora $\vec{r}$ na tretiu.

Rovnicu si sformulujeme v *Pythone*, pričom premennou bude polohový vektor Zeme $\vec{r}$:

```
def gravity(r):
    """
    Parameters
    ----------
    r : pole
        okamzita poloha telesa vzhladom ku Slnku

    Returns
    -------
    acceleration : pole
        okamzity vektor zrychlenia telesa vzhladom ku Slnku

    """
    # Konstanty
    G = 6.674e-11        # Gravitacna konstanta [m^3 / kg s^2]
    m_sun = 1.989e30     # Hmotnost Slnka [kg]

    # Pohybova rovnica
    rmag = np.linalg.norm(r)                # velkost vektora r
    acceleration =(-G*m_sun*r)/(rmag**3)    # gravitacne zrychlenie

    return acceleration
```

Toto je teda pohybová rovnica, ktorú chceme riešiť numericky. Než sa pustíme do jej integrovania, vytvoríme si prázdne polia, do ktorých budeme ukladať výsledky integrácie, zadáme počiatočné podmienky a vlastnosti simulácie. Povedzme, že chceme integrovať pohyb Zeme počas jedného siderického roka (31556926 s) s časovým krokom 1000:

```
dt = 1000           # casovy krok [s]
tf = 32000000       # Konecny cas - cca 1 siderický rok [s]
N = int(tf/dt)      # Pocet krokov
```

Zaujímať nás bude poloha a rýchlosť Zeme v čase t počas celej integrácie. Vytvoríme si preto tri prázdne (nulové) *numpy* polia. (Na zápis dát môžete využívať aj zoznamy - lists alebo si ich vypisovať a čítať zo súboru). Polia pre polohu a rýchlosť musia mať tri stĺpce pre tri zložky vektorov. Dĺžky polí musia byť rovné počtu krokov numerickej integrácie:

```
t = np.zeros(N)             # jednorozmerne pole pre cas, dlzka N
pos = np.zeros((len(t),3))  # trojrozmerne pole pre polohovy vektor
vel = np.zeros((len(t),3))  # trojrozmerne pole pre rychlostny vektor
```

Následne si zadefinuje počiatočné podmienky sústavy, ktoré sú: poloha a rýchlosť Zeme v počiatočnom čase. Počiatočné podmienky si vložíme do prvých riadkov vytvorených polí:

```
t[0] = 0.                               # t(0)= 0 s
pos[0,:] = np.array([149.6e9,0,0])      # pol. vek. Zeme v t(0): x=1 AU, y=0, z=0
vel[0,:] = np.array([0,29e3,0])         # rych. vek. Zeme v t(0):
                                        # x'=0, y'=29e3 m/s, z'=0
```

Cvičenie 3

A teraz môžete prikročiť k napísaniu vlastného simulátora pohybovej rovnice. Ako integrátor budeme využívať metódu Runge-Kutta štvrtého rádu. Vstupom do metódy budú tri vytvorené polia, ktoré budeme napĺnať výsledkami integrácie, počet krokov integrácie a ako parameter vstupuje aj metóda f, ktorá je našou pohybovou rovnicou. Pri metóde RK4 si najskôr vyrátame štyri medzikroky a následne určíme celkové zmeny polohy, rýchlosti a času po jednom kroku:

```
def sim(array_pos,array_vel,array_t,n,f):
    """
    Parameters
    ----------
    array_pos : pole
        3D pole, do ktoreho budeme pridavat hodnoty polohoveho vektora v
        jednotlivych krokoch. Prvy element pola musi obsahovat
        pociatocnu polohu.
    array_vel : pole
        3D pole, do ktoreho budeme pridavat hodnoty rychlostneho vektora v
        jednotlivych krokoch. Prvy element pola musi obsahovat
        pociatocnu rychlost.
    array_t : pole
        1D pole, do ktoreho budeme pridavat hodnoty casu v
        jednotlivych krokoch. Prvy element pola musi obsahovat
        pociatocny cas.
    n : integer
        pocet krokov integracie
    f : method
      pohybova rovnica na urcenie zrychlenia

    Returns
    -------
    Metoda vrati naplnene polia.

    """
    for i in range(1,n):
        """
        Metoda Runge-Kutta stvrteho radu.
        """
        # Prvy medzikrok
        pos1 = array_pos[i-1,:]
        vel1 = array_vel[i-1,:]
        acc1 = f(pos1)

        # Druhy medzikrok
        pos2 = array_pos[i-1,:] + 0.5*vel1*dt
        vel2 = array_vel[i-1,:] + 0.5*acc1*dt
        acc2 = f(pos2)

        # Treti medzikrok
        pos3 = array_pos[i-1,:] + 0.5*vel2*dt
        vel3 = array_vel[i-1,:] + 0.5*acc2*dt
        acc3 = f(pos3)

        # stvrty medzikrok
        pos4 = array_pos[i-1,:] + vel3*dt
        vel4 = array_vel[i-1,:] + acc3*dt
        acc4 = f(pos4)

        # Celkova zmena v parameterov po jednom kroku
        array_pos[i] = array_pos[i-1,:] + (dt/6.0)*(vel1+2*vel2+2*vel3+vel4)
        array_vel[i] = array_vel[i-1,:] + (dt/6.0)*(acc1+2*acc2+2*acc3+acc4)
```

```
54          array_t[i] = dt*i
55
56      return
```

Táto metóda nám vráti naplnené polia polohy, rýchlosti a času. Simulovanie pohybu je vlastne týmto krokom dokončené. Aby sme si naše výsledky aj vykreslili, vytvoríme jednoduchú metódu na vytvorenie (v našom prípade 3D) grafu. Vstupom do metódy bude pole polôh telesa:

```
1  def plot(array):
2      """
3      Parameters
4      ----------
5      array : pole
6          3D pole, ktore chceme vykreslovat na grafe
7
8      Returns
9      -------
10     Zobrazenie trajektorie telesa v 3D.
11
12     """
13     import matplotlib.pyplot as plt
14     import mpl_toolkits.mplot3d as plt3d
15     from mpl_toolkits.mplot3d import Axes3D
16     fig = plt.figure()
17     ax = fig.add_subplot(111, projection='3d')
18     ax.set_aspect("equal")
19     ax.scatter(0,0,0,'r.') # poloha Slnka
20     ax.plot(array[:,0],array[:,1],array[:,2], '-r')
21     plt.show()
22
23     return
```

Týmto máme náš simulátor pohybu hotový. Spustíme ho zavolaním metód *sim* a *plot*.

```
1 sim(pos,vel,t,N)
2 plot(pos)
```