Tartalomjegyzék

Feladatok

Bevezet?

- Írjunk egy **fixed_point** nev? függvényt, ami két paramétert kap: egy **f** függvényt és egy **x0** értéket! Az **f**-et iteratívan hattatva az **x0** kiindulópontból keressük meg a függvény fixpontját. Álljunk meg, ha a lépés kisebb mint 0.00001.
  - Oldjuk meg, hogy legyen egy **tol** opcionális paramétere a függvénynek, 0.00001 alapértelmezett értékkel. Ezzel a megállási pontosságot lehessen megadni!
  - Adjunk hozzá egy **maxiter** opcionális paramétert (default: 200), amivel limitálhatjuk az iterációk számát!

Ferde hajítás

Szimuláljuk a ferde hajítást közegellenállással. A közegellenállási er? mindig a sebességgel ellentéte irányú, $F_k = k \cdot v^2$, ahol a *k* egy összetett paraméter, de ezzel most nem tör?dünk, csak állítgatjuk. A test kezdetben az x_0 = (0,0) pontban van, sebessége pedig *v*0 = (v_x,v_y) (ezt mi adjuk majg meg lehet változtatni). Legyen egy epszilon változónk, mondjuk 0.01, ez lesz a lépéköz. Minden lépésben $\epsilon \cdot v$ vektorral elmozdítjuk a testet, majd a sebességét csökkentjük $\epsilon \cdot (F_k + g)$-vel (*g* a gravitációs gyorsulás, konstans [0,9.81] vektor). Tároljuk ez a mozgás pontjait addig, amíg újra nem ér földet a test és rajzoljuk ki!

Fraktál fa

- Töltsük le a fractree.py file-t és futtassuk le!
- Próbáljuk meg nagyvonalakban megérteni a kódot, nem kell érteni minden számolást, csak azt, hogy kb hol mi történik.

- Módosítsuk a kódot, hogy többször ágazzon el (mondjuk 5 jó szám, annál többel már lehet lassú lesz).
- Elég uncsi, hogy mindig ugyanazt a fát rajzolja ki. Módosítuk, hogy az elágazás szöge (**branchAngle**) véletlen szám legyen **[0, pi / 2]** intervallumon.
- Csináljuk meg ugyanezt a törzs arányával is (**trunkRatio**), legyen mondjuk **[0.25, 0.75]** intervallumon véletlen szám!

- Elég uncsi még mindig, mert még mindig nagyon szimmetrikus, érjük most el, hogy a két ág más szög szerint ágazzon el:
    - Csináljunk a **branchAngle** helyett **branchAngleA** és **branchAngleB** változókat, legyenek randomok mint a **branchAngle** volt.
    - Cseréljük le a **tree** függvényben a **pB** és **pC** pontok létrehozásakor a **branchAngle**-t, az újonnan létrehozottakra, de az egyiknél az egyikre, másiknál a másikra!

- Az utolsó probléma már csak az, hogy még így is túl szabályosnak tûnik, mert a szögek állandóak, érjük el, hogy ágazás közben változzanak:
    - A **tree** függvény végén növeljük a **branchAngleA**, **branchAngleB** és **trunkRatio** változókat egy véletlen számmal, mondjuk **[-0.02, 0.02]** közöttivel.
    - Ahhoz, hogy ez mûködjön globálissá kell tenni a függvényben ezeket a változókat, szóval írjuk be a **tree** függvény elejére, hogy:

```
def tree(p0, p1, limit):
    global branchAngleA
    global branchAngleB
    global trunkRatio
    ...
```

- Próbáljunk olyan beállításokat találni, ami nekünk szimpatikus fát rajzol!

Sudoku

Megoldunk egy sudokut. El?ször a lenti kód kell.

- Egészítsük ki a check_board füvvgényt, hogy ellen?rizze, hogy a teljes tábla ki van-e töltve helyesen! Használhatjuk a check_value függvényt, ami megnézi, hogy az n szám jó-e az i,j pozícióba.
- Oldjuk meg a sudokut backtrack algoritmussal! Írjunk egy solve_sudoku függvényt, ami True-val tér vissza, ha ki van töltve a tábla, False-szal pedig, ha nem kitölthetetlen! Menjünk végig a függvényben a tábla sorain és ha valahol 0 van, ott kezdjük el próbálgatni az értékeket. Ha egy érték beleillik, akkor az új (az értékkel kitöltött) táblára hívjuk meg rekurzívan a függvényt, ami megmondja, hogy kitölthet?-e az így keletkezett tábla, ha igen, akkor a tábla ki lesz töltve, ha nem, töröljük ki az értéket, amit beírtunk!

```
def check_value(board, i, j):
    n = board[i,j]
    square = board[i/3*3:i/3*3+3,j/3*3:j/3*3+3].ravel().tolist()
    row = board[i,:].ravel().tolist()
    column = board[:,j].ravel().tolist()
    square_ok = square.count(n) < 2
    row_ok = row.count(n) < 2
    column_ok = column.count(n) < 2
    return square_ok and row_ok and column_ok
def check_board(board):
    pass
def display(board):
    for i in range(9):
        if i in [3, 6]:
            print '------+-------+------'
        for j in range(9):
```

```
            if j in [3, 6]:
                print '|',
            print board[i,j],
        print
board = np.array([[0, 1, 7, 0, 9, 8, 2, 3, 4],
                  [0, 8, 0, 1, 3, 0, 7, 0, 6],
                  [3, 0, 6, 2, 7, 5, 8, 9, 1],
                  [6, 0, 2, 8, 4, 9, 0, 1, 5],
                  [1, 3, 0, 5, 2, 6, 0, 0, 7],
                  [9, 5, 4, 0, 1, 3, 6, 8, 2],
                  [4, 9, 5, 0, 6, 2, 1, 0, 8],
                  [7, 2, 0, 4, 8, 1, 5, 0, 9],
                  [8, 6, 1, 0, 5, 0, 4, 0, 0]])
```