PYTHON

Geometrija 1

Python urejevalnik se nahaja na strani:
`https://www.w3resource.com/python-exercises/python-basic-exercises.php#EDITOR`

Projektivna geometrija

Premica

Enačba premice kot jo poznamo iz šole je $y = k\,x + n$. Tej obliki enačbe pravimo eksplicitna oblika. Eksplicitna enačba ne opisuje vseh premic. Premice ki so pravokotne na os x ne moremo zapisati z eksplicitno enačbo. Implicitna oblika enačbe je naslednja $a\,x + b\,y + c = 0$. Ta pa lahko zapiše vsako premico. Zapis ni enoličen, ker enačba pomnožena s poljubnim faktorjem različnim od nič predstavlja isto premico.

Torej s trojico realnih števil (a, b, c) je premica enolično določena. Ne velja pa obratno, kot smo videli, vsaka trojica $(\lambda\,a, \lambda\,b, \lambda\,c)$ določa isto premico za poljuben $\lambda \neq 0$.

V primeru, ko sta prvi dve komponenti enaki nič trojica $(0, 0, c)$ ne predstavlja premice.

Točka

Točka v ravnini je enolično določena s parom koordinat (x, y). Točka (x, y) leži na premici (a, b, c), če je $a\,x + b\,y + c = 0$.

Točka in premica v projektivni geometriji

Tu bomo našli posplošitev na neskončne točke in neskončne premice.

V projektivni ravnini nastopata točka in premica simetrično. Tudi točka je v projektivni ravnini predstavljena s trojico realnih števil (x, y, z). Če trojico števil pomnožimo s poljubnim faktorjem različnim od 0, bo ta trojica predstavljala isto točko. Kartezične koordinate točke, predstavljene s trojico (x, y, z), dobimo tako, da trojico delimo z zadnjo komponento z, torej $(x/z, y/z, 1)$ prvi dve komponenti sta abscisa in ordinata točke. Trojica, kjer je tretja komponenta enaka nič ne predstavlja točke. Vendar pa lahko posplošimo pojem točke in dodamo tudi neskončne točke. Neskončne točke pripadajo neskončni premici, $0 \cdot x + 0 \cdot y + 1 \cdot 0 = 0$

Trojica, kjer so vse tri komponente enake 0 ne predstavlja objekta projektivna ravnine.

1. **Premica skozi dve točki** Par različnih točk $t_1 = (x_1, y_1, 1)$ in $t_2 = (x_2, y_2, 1)$ enolično določa premico, ki poteka skozi ti dve točki. Določi trojico (a, b, c) v odvisnosti od T_1 in T_2.

   Velja:

   $$a\,x_i + b\,y_i + c = 0, \quad i = 1, 2$$

   Premica je usmerjena proti neskončni točki $t_\infty = t_2 - t_1 = (x_2 - x_1, y_2 - y_1, 0)$. Če vzamemo, da je $a = -(y_2 - y_1)$ in $b = (x_2 - x_1)$ in vstavimo $x = x_i$ in $y = y_i$, $i = 1, 2$, dobimo v obeh primerih enako vrednost za c.

   $$(y_2 - y_1)\,x_1 + (x_2 - x_1)y_1 + c = 0, \quad -y_2x_1 + y_1x_1 + x_2y_1 - x_1y_1 + c = 0, \quad c = x_2y_1 - x_1y_2.$$

   Od tod sledi, da obe točki ležita na premici, če je premica predstavljena s trojico:

   $$l = (y_1 - y_2, x_2 - x_1, x_2y_1 - x_1y_2).$$

Prvi dve komponenti trojice, ki pripadata premici sta $(y_1 - y_2, x_2 - x_1)$ in predstavljata smer, ki je pravokotna na smer premice. Imenujemo ga normalni vektor premice. Če gremo od točke T_1 proti točki T_2 kaže normalni vektor na levi breg premice.

Skalarni produkt komponent točke $p = (x_0, y_0, 1)$ s trojico, ki predstavlja premico $l = (a, b, c)$, $p \cdot l = ax_0 + by_0 + c$ je pozitiven, če leži točka na istem bregu premice, kamor kaže normalni vektor in je negativen, če leži točka na drugi starani. V primeru, ko je $a^2 + b^2 = 1$, je vrednost produkta enaka razdalji točke od premice.

2. **Točka presečišče dveh premic** Enako lahko določimo presečno točko premic, kjer v zgornji enačbi zamenjamo vlogo točk in premic.

$$l_i = (a_i, b_i, c_i), \quad i = 1, 2, \quad p = (b_1c_2 - b_2c_1, a_2c_1 - a_1c_2, a_1b_2 - a_2b_1)$$

3. **Operacije** V množici premic in točk definiramo vsoto, produkt s številom in dva produkta. Vsoto in produkt s številom definiramo na običajen način po komponentah.
   Produkta sta:
   - Skalarni produkt $(l_1, m_1, n_1) \cdot (l_2, m_2, n_2) = l_1l_2 + m_1m_2 + n_1n_2$.
   - Vektorski produkt $(l_1, m_1, n_1) \times (l_2, m_2, n_2) = (m_1n_2 - m_2n_1, l_2n_1 - l_1n_2, l_1m_2 - l_2m_1)$

Rešitve 1

Graphics

```
#!/usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt
def bounding_box(x0, x1, y0, y1):
    return np.array([[0, 1, y0], [0, -1, y1], [1, 0, x0], [-1, 0, x1]])
def triangle(a, b, c):
    return np.linalg.det([a, b, c])
def inside_bb(x):
    return all([np.dot(y,x) >=0 for y in bb])
def pts_lin(ln):
    clipl = set()
    for x in bb:
        y = np.cross(ln,x)
        if abs(y[-1]) > 1e-5:
            y = y/y[-1]
        if inside_bb(y):
            clipl.add(tuple(y))
    return clipl
def graphics(x0, x1, y0, y1):
    global bb
    bb = bounding_box(x0, x1, y0, y1)
    plt.axes().set_aspect('equal')
    plt.xlim(x0, x1)
    plt.ylim(y0, y1)
def plt_lin(ln):
    clp = pts_lin(ln)
    plt.plot([x[0] for x in clp],[x[1] for x in clp])
def _pts(pts):
    x, y = [], []
    for pt in pts:
        pt = pt/pt[-1]
        if inside_bb(pt):
            x.append(pt[0])
            y.append(pt[1])
    return x, y
def plt_pts(pts):
    x, y = _pts(pts)
    plt.scatter(x, y)
def plt_ply(pts):
    x, y = _pts(pts)
    plt.plot(x, y)
```

Plane objects

```python
#!/usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt
from plane_graph import *
#--------------------- plane objects --------------
class Pobj:
    def __init__(self, vector):
        if len(vector) == 3:
            self.vector = np.array(vector)
        elif len(vector) == 2:
            self.vector = np.append(vector, 1)
    def __add__(self, other):
        return Pobj(self.vector + other.vector)
    def __sub__(self, other):
        return Pobj(self.vector - other.vector)
    def __mul__(self, other):
        if type(other) == Pobj:
            return np.dot(self.vector, other.vector)
        elif type(other) == float or type(other) == int:
            return Pobj(self.vector * other)
    def __rmul__(self, other):
        if type(other) == float or type(other) == int:
            return self * other
    def __truediv__(self, other):
        if type(other) == float or type(other) == int:
            return Pobj(self.vector / other)
    def __xor__(self, other):
        return Pobj(np.cross(self.vector, other.vector))
    def __str__(self):
        return '(%0.2f, %0.2f, %0.2f)' % tuple(self.vector)
    def normalize(self, pn = None):
        if pn == 'p':
            c = self.vector[-1]
            self.vector = self.vector/c
        elif pn == 'l':
            v = self.vector[:-1]
            n = np.sqrt(np.dot(v, v))
            self.vector = self.vector/n
        elif pn == 'v':
            v = self.vector[:-1]
            n = np.sqrt(np.dot(v, v))
            v = v / n
            self.vector = v + [0]
        else:
            m = np.max([abs(x) for x in self.vector])
            self.vector = self.vector/m
```

Graphics

```
# -------------------- graphics ---------------------------------
def plt_obj(obj, poly = False):
    if type(obj) == list:
        if poly:
            plt_ply([x.vector for x in obj])
        else:
            plt_pts([x.vector for x in obj])
    else:
        plt_lin(obj.vector)
```

Test

```
# -------------------- test -------------------------------------
if __name__ == '__main__':
    graphics(0, 10, 0, 10)
    pt = Pobj((3,2))
    plt_obj([pt])
    line = (2, -3, 7)
    ln = Pobj(line)
    plt_obj(ln)
    pi = ln
    pi.normalize("v")
    li = pi^pt
    plt_obj(li)
    plt.show()
```