Poročilo projekta pri predmetu Matematično modeliranje

Evgenija Burger

22. avgust 2021

Izbrala sem lažji projekt.

1 Navodilo projekta

Napišite program, ki simulira gibanje točke po Bezierovi krivulji stopnje n. Vzporedno s to animacijo naj program izrisuje ukrivljenost krivulje. Podatki so kontrolne točke in korak parametra.

2 Seznam funkcij s kratkimi opisi

Pri reševanju naloge sem uporabila nekaj osnovnih funkcij, ki smo jih spoznali na vajah:

- `deCasteljau(b, t)` - funkcija, ki temelji na de Casteljauvemu algoritmu in vrne točko ob času t na krivulji podani s kontrolnimi točkami.
- `plotBezier(b)` - funkcija, ki izriše Bezierjevo krivuljo in podane kontrolne točke.
- `bezier_der(b, t, risanje)` - izračuna tangentni vektor na krivuljo pri izbranem parametru.

Preostale funkcije sem sestavila sama s pomočjo znanja, ki sem ga pridobila na vajah, ali pa s pomočjo informacij na spletu. Glavne funkcije, ki sem jih sestavila za rešitev problema so naslednje:

- `ukrivljenost(b, T)` - funkcija, ki sprejme kontrolne točke krivulje in seznam parametrov, pri katerih želimo izračunati ukrivljenost krivulje. Vrne seznam vrednosti ukrivljenosti v podanih točkah.
- `simulacija_gibanja(b, korak, pot)` - funkcija simulira gibanje delca po krivulji in sproti izrisuje ukrivljenost krivulje. Če je število korakov simulacije majhno, lahko 'vklopimo' možnost *pot*, ki na krivulji izriše pot delca v obliki daljic.

- `narisi_daljico(T1, T2)` - nariše daljico med podanima točkama, kar uporabimo za risanje poti delca.

Vemo, da ima ukrivljenost tudi geometrijski pomen - obratna vrednost ukrivljenosti predstavlja polmer pritisnjene krožnice. Zato sem dodala nekaj funkcij, ki rešitev predstavijo še nekoliko drugače - ukrivljenost v točki lahko predstavimo tudi s pritisnjeno krožnico. Funkcije, ki sem jih sestavila v ta namen so:

- `pritisnjena_kroznica(b, z, bezier)` - funkcija, ki pri parametru z nariše pritisnjeno krožnico na krivuljo. Vhodni podatek *bezier* 'vklopi' risanje celotne krivulje, kar je priročno, ko želimo funkcijo uporabiti samo, ni pa praktično, kadar jo uporabljamo znotraj ostalih funkcij, kot v našem primeru.

- `normalni_vektor(b, T)` - funckija izračuna normalne vektorje v točkah, podanih v seznamu T.

- `simulacija_kroznice(b, korak, pot)` - funkcija simulira gibanje delca po krivulji, podobno kot funkcija `simulacija_gibanja`, sproti pa v vsaki točki izriše pritisnjeno krožnico na ravnino. Prav tako sproti izrisuje ukrivljenost krivulje, iz česar se lepo vidi vpliv ukrivljenosti na velikost pritisnjene krožnice.

V datoteki `demo` se nahaja nekaj primerov krivulj na katerih lahko uporabimo opisane funkcije pri različnem številu korakov.

V mapi se nahaja tudi funkcija `parametricno(b)`. Sprva sem se reševanja lotila z njeno pomočjo, vendar sem ugotovila, da ta način pri velikem številu korakov ni učinkovit. Več o tem sem zapisala v komentarju poročila.

3 Podrobnejši opis načina reševanja

3.1 Osnovna rešitev

Osnovna rešitev problema je torej predstavljena v funckiji `simulacija_gibanja`. Funkcija sprejme kontrolne točke krivulje in število korakov simulacije, prav tako pa se mi je zdelo pomembno dodati možnost, s katero lahko vklopimo ali izklopimo risanje opravljene poti delca na krivulji. To storimo tako, da kot tretji vhodni podatek *pot* podamo 0 (izklop) ali 1 (vklop).

Opisana možnost ima smisel pri majhnem številu korakov simulacije (na primer pri 10 ali 5 korakih), saj takrat delec veliko vmesnih točk na krivulji izpusti. Njegova pot je tako skupek daljic, ki povezujejo točke na krivulji, v katerih se delec ustavi. Tem večje je število korakov, bolj postane izrisana pot podobna krivulji. Funkcija `plotBezier`, ki smo jo obravnavali na vajah, pri risanju krivulje razdeli interval $[0, 1]$ za parameter t na 100 enakih delov,

tako se tudi opisani izris poti pri 100 korakih simulacije ujema z izrisano Bezierjevo krivuljo.

Funckija v prvem koraku nariše celotno podano Bezierjevo krivuljo s pomočjo funkcije `plotBezier`. Nato interval $[0, 1]$ razdeli na enake dele, katerih je toliko, kot je število podanih korakov simulacije. Deli predstavljajo 'čase' ob katerih funkcija izriše delec na krivulji. Do točke na krivulji pridemo s pomočjo funckije `deCasteljau`. V kolikor smo vklopili možnost risanja poti, bo na vsakem koraku izrisala še daljico med prejšnjo točko in točko v kateri se nahajamo ob danem času.

Poleg animacije potovanja delca po krivulji funkcija vzporedno izrisuje ukrivljenost krivulje. Ukrivljenost krivulje lahko izračunamo po enačbi

$$\kappa = \frac{\|\dot{\vec{r}} \times \ddot{\vec{r}}\|}{\|\dot{\vec{r}}\|}.$$

Ker imamo opravka z ravninsko krivuljo, bosta prva in druga komponenta pri izračunanem vektorskem produktu enaki 0, zato lahko v funkciji `ukrivljenost` to izkoristimo in vzamemo le tretjo komponento vektorja. Prav tako lahko na tem mestu izpustimo absolutno vrednost in dobimo predznačeno ukrivljenost.

Ukrivljenost krivulje izrisujemo sproti, če pa želimo, da se nam graf spreminjanja ukrivljenosti izriše že na začetku, odkomentiramo ukaz `plot(T, ukrivljenost(b, T));`.

Ko sem se odločala na kakšen način želim izrisati graf ukrivljenosti sem imela dve možnosti:

1. Graf prikažem kot funkcijo, sestavljeno iz linearnih funkcij na podintervalih, ki jih določajo delilne točke. Linearne funkcije povezujejo vrednosti ukrivljenosti v delilnih točkah oziroma v vsakem koraku simulacije.

2. Ne glede na število korakov simulacije izrišem graf ukrivljenosti čim bolj natančno. To pomeni, da v vsakem primeru graf predstavim v odvisnosti od 100 ekvidistantnih točk na intervalu $[0, 1]$.

Odločila sem se za drugo možnost. Če bi funkciji podali majhno število korakov in graf ukrivljenosti izrisali le na podlagi teh točk (kot opisuje prva možnost), bi se lahko zgodilo, da graf ne bi bil realen - v primeru 'ostrega zavoja' krivulje imamo lahko zelo veliko ukrivljenost, vendar bo morda delec velik del zavoja preskočil in se visoka vrednost ukrivljenosti na grafu ne bo zabeležila. Zato na vsakem koraku simulacije vmesni interval razdelimo na dele velikosti $\frac{1}{100}$ (seveda le v primeru, ko je število podanih korakov manjše od 100) in izrišemo graf ukrivljenosti v vseh vmesnih točkah.

3.2 Dodatek

Iz Analize 2b se spomnimo, da ima ukrivljenost tudi geometrijski pomen. Obratna vrednost predstavlja polmer pritisnjene krožnice na krivuljo.

$$\kappa = \frac{1}{R}$$

To je krožnica, ki gre skozi dano točko in se krivulji najbolje prilega. Njeno središče leži v smeri normalnega vektorja na krivuljo. Pri tem je pomembno, da upoštevamo smer normalnega vektorja, saj ni vseeno, na kateri strani krivulje leži krožnica.

Normalni vektor izračunamo po naslednji formuli:

$$\vec{N} = \frac{\dot{\vec{r}} \times (\ddot{\vec{r}} \times \dot{\vec{r}})}{\|\dot{\vec{r}} \times (\ddot{\vec{r}} \times \dot{\vec{r}})\|}$$

S funkcijo `normalni_vektor` izračunamo normiran normalni vektor na krivuljo pri izbranem parametru. Izračunan vektor nato uporabimo pri funkciji `pritisnjena_krožnica`, ki v dani točki izriše pritisnjeno krožnico na krivuljo, prav tako pa nariše vektor, ki povezuje središče krožnice in točko na krivulji. Vse skupaj uporabimo pri `simulacija_kroznice`, ki deluje na zelo podoben način kot funkcija `simulacija_gibanja`, le da v vsaki točki izriše tudi pritisnjeno krožnico na krivuljo.

Prav tako sproti izrisuje ukrivljenost krivulje, iz česar se lepo vidi, kako majhna ukrivljenost pomeni veliko krožnico in obratno.

4 Komentar k rešitvi

Reševanja problema ukrivljenosti in iskanja normalnega vektorja sem se najprej lotila tako, da sem Bezierjevo krivuljo predstavila parametrično

$$b(t) = (x(t), y(t), 0)$$

s pomočjo funkcije `parametricno(b)`, ki se nahaja v mapi z ostalimi funkcijami. Ker imamo opravka z ravninsko parametrično krivuljo lahko ukrivljenosti in normalni vektor izračunamo nekoliko drugače, kar sem sprva poskušala tudi sama. Enačba se glasi:

$$\kappa = \frac{|\dot{x}\ddot{y} - \ddot{x}\dot{y}|}{(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}}$$

Začela sem s parametrizacijo krivulje in odvajanjem obeh komponentnih funkcij, ki sem jih nato vstavila v zgornjo enačbo. Tako sem definirala tudi normalni vektor. To sem naredila na spodaj prikazan način.

```
syms s
[x, y] = parametricno(b);
xx = x(s); yy = y(s);
dx1 = diff(xx); dy1 = diff(yy);
dx = @(t) vpa(subs(dx1, s, t)); dy = @(t) vpa(subs(dy1, s, t));
ddx1 = diff(xx, 2); ddy1 = diff(yy, 2);
ddx = @(t) vpa(subs(ddx1, s, t)); ddy = @(t) vpa(subs(ddy1, s, t));

kappa = @(t) (dx(t)*ddy(t) - dy(t)*ddx(t))/(dx(t)^2 + dy(t)^2)^(3/2);
normalni_vektor = @(t) [dy(t)*(ddx(t)*dy(t)-dx(t)*ddy(t));
-dx(t)*(ddx(t)*dy(t)-dx(t)*ddy(t))];
```

Na začetku sem definirala funkciji `normalni_vektor2` in `ukrivljenostSez`, ki sta temeljili na zgoraj opisanem postopku, imeli pa sta enako vlogo kot končni funkciji `normalni_vektor` in `ukrivljenost`. Ugotovila sem, da simulacija traja zelo dolgo, zato ju nisem uporabila. Kot zanimivost sem primerjala čase, ki so jih funkcije potrebovale, kar je prikazano spodaj. Končni funkciji za izračun ukrivljenosti in normalnega vektorja sta občutno hitrejši.

```
b1 = [0, 1, 2, 4; 0, -1, -1, 2];
tic
ukrivljenost(b1, 0.5); \% Elapsed time is 0.003240 seconds.
toc
tic
ukrivljenostSez(b1, 0.5); \% Elapsed time is 0.605890 seconds.
toc
tic
normalni_vektor(b1, [0.1, 0.5, 0.7]); \% Elapsed time is 0.127684 seconds.
toc
tic
normalni_vektor2(b1, [0.1, 0.5, 0.7]);\% Elapsed time is 2.167341 seconds.
toc
```

5 Viri in literatura

- Zapiski s predavanj in vaj pri predmetu Matematično modeliranje v študijskem letu 2020/21, gradivo dostopno na spletni učilnici FMF študijskega leta 2020/21.
- Josip Globevnik, Miha Brojan, Analiza II, dostopno na https://www.fmf.uni-lj.si/ globevnik/skriptaII.pdf.
- Zapiski s predavanj pri predmetu Analiza 2b v študijskem letu 2019/20