Roko Brlek

14.6.2022

Simuliranje jednostavnih kolonija mrava koristeći diferencijalne jednadžbe.

Uvod

Cilj projekta „Simuliranje jednostavnih kolonija mrava koristeći diferencijalne jednadžbe“ je izrađivanje jednostavnog matematičkog modela kolonije mrava sa dvije varijable te prikaz rezultata na interaktivan način.

Matematički model

Matematički model se sastoji od dvije glavne diferencijalne jednadžbe (Jednadžba 1 i Jednadžba 2)

Jednadžba

Jednadžba

Gdje su i karakteristična količina hrane i karakteristična veličina kolonije. Varijable se zovu „karakteristične“ jer mogu biti negativne, a njihov predznak ima važno značenje u kontekstu kolonije mrava.

Pošto ja nemam dovoljno matematičkog znanja da nađem analitičko rješenje jednadžbi, u ovom radu koristim metodu prosječnog određivanja rješenja koja aproksimira funkcijske promjene u vremenu koristeći derivacije.

Ova metoda se često koristi kada diferencijalna jednadžba nema poznatog analitičkog rješenja, poput Jednadžba 3

Jednadžba

Prikaz podataka

Koristeći metodu opisanu prije program sprema funkcijsku vrijednost svakog nakon svakog intervala .

Ti se podatci pokazuju u faznom prostoru kolonije koristeći Python library [Matplotlib](https://matplotlib.org/stable/index.html), koja služi za prikazivanje podataka na različite načine.

Interaktivan dio prikaza podataka se isto programira uz pomoć Matplotlib-a. U interaktivnom grafu se mogu mijenjati koeficijenti (Jednadžba 1) i (Jednadžba 2), interval vremena te početne vrijednosti . Uz mogućnost mijenjanja tih podataka postoji i gumb koji ih resetira na početne vrijednosti koje su unese u programu.

Uz prikaz podatak je napravljeno vektorsko polje koje prikazuje derivaciju svake točke kao vektor.

Matematički model

Matematički model se sastoji od dvije glavne jednadžbe (Jednadžba 1 i Jednadžba 2)

Gdje su i karakteristična količina hrane i karakteristična veličina kolonije. Varijable se zovu „karakteristične“ jer mogu biti negativne, a njihov predznak ima važno značenje u kontekstu kolonije mrava.

Karakteristike cikličnog procesa

Jednadžba 2 kaže da će brzina rasta karakteristične veličine kolonije biti linearno ovisna o karakterističnoj količini hrane, a Jednadžba 1 kaže da će brzina rasta karakteristične količine hrane biti linearno ovisna o negativnoj vrijednosti karakteristične veličine kolonije. To jest: što je više hrane to će kolonija biti veća, te što je kolonija veća to je manje hrane. Kada je negativna to znači da ima previše mrava da bi kolonija mogla rasti, to jest da ima nedovoljno hrane te da će padati. Kada je negativna to znači da ima dovoljno malo mrava da se može povećavati tj. oporaviti.

Ove jednadžbe rezultiraju u cikličnom procesu koji je prikazan dolje.(Slika 1)

Slika

Da bi objasnili proces pretpostavit ćemo da počinjemo u stanju za koje vrijedi . Zbog predznaka tih početnih vrijednosti će kroz vrijeme padati jer ima previše mrava da bi se hrana oporavila, a će kroz vrijeme rasti jer ima dovoljno hrane da bi kolonija rasla.To će vrijediti sve dok ne dođemo do stanja za koje vrijedi . Činjenicu da je tumačimo kao da su mravi potrošili toliko puno hrane da je nema dovoljno da se podržava rast kolonije te zato veličina kolonije pada. U trećoj fazi procesa vrijedi , to tumačimo kao da je veličina kolonije pala dovoljno puno da ima toliko malo mrava da se količina hrane može početi obnavljati, te zato količina hrane raste u toj fazi. No pošto još uvijek nema dovoljno hrane da bi kolonija mrava mogla rasti kolinja se još uvijek smanjuje. U četvrtoj fazi procesa vrijedi , što tumačimo kao da je hrana rasla dovoljno dugo da je opet ima dovoljno da bi kolonija mogla rasti, te kao posljedica toga kolonija raste. No hrana još uvijek raste jer ima premalo mrava da bi se trošila brže nego što se obnavlja. Nakon četvrte faze se vraćamo u prvu fazu gdje je , te se tako proces ponavlja ciklički.

Matematika modela

No nama model ne znači ništa ako nemamo način za dobiti podatke iz njega, to bi se inače napravilo tako da se nađe analitičko rješenje sistema diferencijalnih jednadžbi. (Jednadžba 1 i Jednadžba 2)  
No pošto ja nemam dovoljno matematičkog znanja da bi znao kako to riješiti odlučio sam koristiti metodu za aproksimiranje rješenja. Metoda koju koristim je jedna od metoda koje se na engleskom poznaju kao „[Finite Difference methods](https://en.wikipedia.org/wiki/Finite_difference_method)“. Metoda koristi derivacije da bi se izračuna prosječna promjena funkcijske vrijednosti u nekom intervalu . Ovdje se ne približava nuli nego je neka mala vrijednost (npr. 0.0001). Vrijednost dobivamo iz činjenice da je iz čega slijedi .  
To jest, da bi smo približno izračunali mi ga zapišemo u obliku zbroja početne vrijednosti i promjene funkcijske vrijednosti za svaki tako da vrijedi .   
To možemo u obliku jedne jednadžbe napisat na sljedeći način (Jednadžba 4)

Jednadžba

Pod uvjetom da znamo mi možemo aproksimirati funkcijsku vrijednost za bilo koji .

U našem modelu gdje imamo dvije varijable vrijede ista pravila.

Jednadžba

Jednadžba

No iz sistema diferencijalnih jednadžbi koje definiraju naš model (Jednadžba 1 i Jednadžba 2) znamo da vrijedi:

Iz toga vidimo da ako znamo neke vrijednosti i možemo izračunati i te   
To nam je važno jer možemo pokazati da ako znamo početne vrijednosti i možemo saznati bilo koji te

Jednadžba

Isto vrijedi i za svaki

Jednadžba

Te smo pokazali da možemo aproksimirati bilo koji i kroz ponavljanje procesa prikazanog gore (Jednadžba 7 i Jednadžba 8) te da zato možemo i naći bilo koji te . Važno je spomenuti da će naša aproksimacija i biti točnija što je manji.

Kod i algoritam programa

Programiranje matematičkog modela

Naš model ima par karakterističnih veličina koje trebamo znati da bi smo mogli bilo što izračunati:

* Sistem diferencijalnih jednadžbi (Jednadžba 1 i Jednadžba 2)
* Vremenski interval
* Početne vrijednosti i
* Parametri i (vidi Jednadžba 1 i Jednadžba 2)

Još jedna veličina koju ćemo dodati je završno vrijeme , njega dodajemo da bi smo odredili kada prestati računati vrijednosti i kroz proces opisan gore (Jednadžba 7 i Jednadžba 8)

Pošto će završni prikaz podataka biti interaktivan moramo naš model programirati tako da može lagano podnijeti promjene u parametrima. Ja sam za taj zadatak izabrao Python klase. Klasa koju koristim da bi programirao model ću odsad nadalje zvati „klasa modela“.  
Klasa modela ima unutarnje stanje koje sadrži sve veličine koje smo prije naveli. Da bi smo te veličine mijenjali koristimo funkciju klase „setState“ koja ažurira unutarnje stanje te klase. Ta funkcija kao parametar prima rječnik sa svim podatcima koje smo prije naveli, a kako bi lakše ažurirali stanje modela imamo funkciju „getState“ koja nam daje baš taj rječnik. Iz toga slijedi da ako želimo ažurirati stanje funkcije prvo zovemo „getState“, mijenjamo rječnik koji nam daje te zovemo funkciju „setState“.

Da bi smo zapravo mogli dobiti podatke funkcijskih vrijednosti našeg modela nakon svakog intervala trebamo napisati funkciju „updateData“ u klasi modela koja će uzeti vrijednosti unutarnjeg stanja modela te izračunati te funkcijske vrijednosti. Te podatke o funkcijskim vrijednostima ja zovem posljedica stanja jer su ti podaci u potpunosti određeni stanjem.

Važna karakteristika naše klase je da funkcija koja ažurira stanje „setState“ ne zove funkciju koja računa podatke „updateData“. Ovo je izabrano zato da bi smo mogli više puta ažurirati unutarnje stanje bez da moramo računati podatke svaki put. To puno ubrzava program jer izbjegava nepotrebni račun.  
Ova kombinacija tri funkcije nam daje ciklus ažuriranja cijele klase koji je prikazan na dijagramu ispod. (Slika 2)

Slika

updateState

getState

updateData

Ovo se jako puno koristi kada u kodu koji radi interaktivni prikaz podataka.

Završne dvije funkcije koje naša klasa modela ima su „getData“ i „logData“. „getData“ je napravljen da bi zapravo mogli dobiti izračunane podatke koji se onda mogu koristiti za razne stvari, a „logData“ uzima podatke te ih zapisuje u file gdje svaka od dvije varijable u našem sistemu diferencijalnih jednadžbi ima svoju oznaku. Ime file-a te koju oznaku koristi koja varijabla su parametri koje korisnik unosi kada zove funkciju.

Programiranje interaktivnog prikaza podatka

Da bi prikazao podatke i napravio taj prikaz interaktivnim koristim Python library „Matplotlib“ koji je napravljen za prikazivanje znanstvenih podataka na različite načine.

Prikaz se sastoji od par glavnih elementa:

* Faznog prostora kolonije mrava sa hranom kao os, te kolonijom kao os. U tom faznom prostoru se nalazi graf podataka koji se dobiju kada se zove funkcija „getData“ klase modela.
* Točka koja predstavlja početne vrijednosti klase modela
* Vektorsko polje koje se prikazuje u faznom prostoru kolonije mrava. Ono se određuje tako da vektor počinje na nekim koordinatama te da je njegov smjer određen vektorom . To jest vektor koji počinje na nekoj točki pokazuje smjer promjene podataka u faznom prostoru kolonije mrava.
* dva klizača čije vrijednosti određuju vrijednosti parametara iz sistema diferencijalnih jednadžbi (Jednadžba 1 i Jednadžba 2)
* klizač čija vrijednost određuje vrijednost intervala vremena
* gumb koji vraća unutarnje stanje klase modela na početno stanje

Sve elemente radimo proceduralno u programu, no da bi ih mogli napraviti interaktivnim radimo novu klasu kojoj prosljeđujemo sve te elemente. Ja ću tu klasu zvati klasa interakcije.

Ta klasa na omogućuje interakciju na par glavnih načina:

* mijenjanje parametara iz sistema diferencijalnih jednadžbi (Jednadžba 1 i Jednadžba 2) tako da mijenjamo vrijednosti na klizačima
* mijenjanje vremenskog intervala tako da mijenjamo vrijednost na klizaču
* mijenjanje početnih vrijednosti klase modela tako da mičemo točku po grafu
* resetiranje unutarnjeg stanja modela kroz gumb

Unutarnje stanje klase interakcije se sastoji od svih elemenata koje smo prije naveli te dvije dodatne varijable. Prva od tih dviju varijabli je važna za kod koji omogućuje mijenjanje početnih vrijednosti klase modela, a druga varijabla je unutarnje stanje klase modela kada smo napravili novi rječnik od klase interakcije, to unutarnje stanje se dobije koristeći „getState“ funkciju klase modela.

Klasa interakcije ima par glavnih funkcija:

* „connect“ funkcija se zove kada se želi omogućiti interakcija sa prikazom podataka
* „updatePlot“ funkcija se zove kada se kroz interakciju sa prikazom podataka promijenilo unutarnje stanje klase modela. Funkcija „updatePlot“ u svom kodu zove funkciju klase modela „updateData“ i „getData“ da ažurira i dobi relevantne podatke te da prema njima ažurira prikaz podataka. Ova funkcija se zove kad god se unutarnje stanje klase modela promijeni.
* „updateQuiver“ je funkcija slična funkciji „updatePlot“, ona se isto tako zove kada se kroz interakciju sa prikazom podataka promijenilo unutarnje stanje klase modela. No ona ne zove ni funkciju „updateData“ ni funkciju „getData“ jer vektorsko polje ovisi samo o unutarnjem stanju klase modela. Ova funkcija se zove kad god se unutarnje stanje klase modela promijeni.
* „onResetButtonClick“ se zove kada korisnik pritisne na gumb koji vraća unutarnje stanje klase modela na ono što je bilo kada se od klase interakcije napravio novi rječnik. Da bi to napravila funkcija koristi jednu od prije spomenutih dodanih varijabli koja skladišti stanje klase modela u trenutku kada smo od klase interakcije napravili novi rječnik. Ona ažurira unutarnje stanje klase modela te zove „updatePlot“ i „updateQuiver“.
* „onSliderUpdate“ se zove kada se vrijednosti klizača promijeni. Ona ažurira unutarnje stanje klase modela te zove „updatePlot“ i „updateQuiver“.
* „onPress“ se zove kada korisnik pritisne na točku . Ta funkcija skladišti podatke o koordinatama miša kada je točka pritisnuta te o koordinatama točke u jednoj od prije spomenutih dodatnih varijabli koju ču ja zvati „pressData“. Važno je spomenuti da ona ne zove „updatePlot“ i „updateQuiver“ jer ne ažurira unutarnje stanje klase modela.
* „onMotion“ se zove samo ako je varijabla „pressData“ nije jednaka „None“ (tj. ako je korisnik kliknuo na točku ) i ako se miš pomaknuo. Ona koristi podatke spremljene u varijabli „pressData“ da izračuna početne vrijednosti . Kada to izračuna ona ažurira unutarnje stanje klase modela te zove funkciju „updatePlot“. Važno je spomenuti Matplotlib detektira pomak miša tako da se i najmanji pomak registrira, te kao posljedica toga mijenjanje početnih vrijednosti . izgleda kao da vučemo točku po koordinatnom sistemu u kojem se nalazi prikaz podataka.
* „onRelease“ se zove kada se otpusti gumb „left click“ na mišu. On promjeni vrijednost varijable „pressData“ u „None“ da bi se „onMotion“ funkcija prestala zvati kada se miš pomakne, te zove funkciju „updatePlot“.

Konfiguracija sustava potreban da bi program radio

Da bi program radio korisnik treba imati instaliran: Python 3.10.0, Matplotlib 3.5.2, Numpy 1.21.4

Program je napravljen na operacijskom sistemu „Windows 10 home“

Backend koji Matplotlib koristi je „TkAgg“.

[Jednadžba 1 1](#_Toc106146774)

[Jednadžba 2 1](#_Toc106146775)

[Jednadžba 3 2](#_Toc106146776)

[Jednadžba 4 4](#_Toc106146777)

[Jednadžba 5 5](#_Toc106146778)

[Jednadžba 6 5](#_Toc106146779)

[Jednadžba 7 5](#_Toc106146780)

[Jednadžba 8 6](#_Toc106146781)

[Slika 1 3](file:///C:\coding\python\matplotlib\ants\documentation.docx#_Toc106146783)

[Slika 2 7](file:///C:\coding\python\matplotlib\ants\documentation.docx#_Toc106146784)