Dynamické modely populací

Robert Mařík

24.03.2023

Přednášky

I Přednášky **5**

1 Funkce a vlastnosti funkcí **7**
1.1 Funkce jedné proměnné . . . 7
1.2 Vlastnosti funkcí . . . 8
1.3 Elementární funkce . . . 9
1.4 Funkce více proměnných a vektorové funkce . . . 13

2 Derivace a integrál **15**
2.1 Průměrná rychlost a okamžitá rychlost . . . 15
2.2 Spojitost . . . 16
2.3 Limita . . . 17
2.4 Definice derivace . . . 17
2.5 Využití derivace . . . 18
2.6 Derivace v populační ekologii . . . 19
2.7 Model tepelné výměny . . . 19
2.8 Analytický výpočet derivace . . . 20
2.9 Numerický výpočet derivace . . . 20
2.10 Integrál a jeho využití . . . 21

3 Diferenciální rovnice **23**
3.1 Obyčejná diferenciální rovnice . . . 23
3.2 Existence a jednoznačnost řešení . . . 24
3.3 Směrové pole . . . 24
3.4 Analytické a numerické řešení . . . 25
3.5 Model ostrovní biogeografie . . . 25
3.6 Transformace diferenciálních rovnic . . . 27

4 Autonomní diferenciální rovnice **29**
4.1 Exponenciální růst . . . 30
4.2 Von Bertalanffyho růst . . . 30
4.3 Logistický růst . . . 30
4.4 Autoregulace a rychlost syntézy proteinů . . . 31

5 Bifurkace **35**
5.1 Bifurkace v logistické diskrétní rovnici . . . 35
5.2 Logistický růst a dvě strategie lovu . . . 36
5.3 Populace pod predačním tlakem . . . 38

6 Lineární algebra, matice **41**
6.1 Vektory . . . 41
6.2 Lineární kombinace . . . 42

6.3 Matice . . . 42
6.4 Jednotková matice a matice inverzní . . . 43
6.5 Matice jako zobrazení . . . 43
6.6 Vlastní čísla a směry . . . 43
6.7 Markovův řetězec, model skladby lesa . . . 43
6.8 Leslieho model růstu populace s věkovou strukturou . . . 45
6.9 Zobecnění Leslieho modelu . . . 45
6.10 Soustavy lineárních rovnic . . . 47
6.11 Lineární autonomní systémy . . . 47

7 Parciální derivace **49**
7.1 Zavedení parciální derivace . . . 49
7.2 Lineární aproximace . . . 50
7.3 Sensitivita a elasticita projekční matice . . . 51
7.4 Rovnice vedení tepla . . . 52
7.5 Fisherova–KPP rovnice . . . 52

8 Autonomní systémy **55**
8.1 Lineární autonomní systémy . . . 55
8.2 Nelineární autonomní systémy . . . 56
8.3 Autonomní systémy v rovině . . . 56
8.4 Lotkův–Volterrův model dravce a kořisti . . . 57
8.5 Volterrův efekt . . . 58

9 Modely konkurence populací **61**
9.1 Konkurence dvou populací . . . 61
9.2 Model konkurence tří druhů . . . 64
9.3 Symbióza dvou druhů . . . 65

10 Modely dravce a kořisti **67**

11 Diskrétní populační modely **69**
11.1 Diskrétní modely jednodruhové populace . . . 69
11.2 Diskrétní modely dvou populací . . . 70

12 Difuzní rovnice **71**

II Cvičení **73**

13 První kroky **75**
13.1 Dva druhy políček . . . 75
13.2 Knihovny . . . 75
13.3 Cykly . . . 75
13.4 Často recyklujeme . . . 76

14 Úvod do jazyka Python **77**
14.1 Ke způsobu práce . . . 77
14.2 K syntaxi v jazyce Python . . . 77
14.3 Aritmetika s čísly . . . 78
14.4 Hrátky s textem . . . 78
14.5 Knihovny . . . 79
14.6 Práce se seznamem hodnot . . . 79
14.7 Práce s poli `np.array` . . . 80
14.8 Tabulky, pandas . . . 81
14.9 Obrázky . . . 82
14.10 Vykreslení dvourozměného pole . . . 83
14.11 Dva obrázky pod sebou, data z tabulky . . . 83
14.12 Růst populace v prostředí s omezenou nosnou kapacitou . . . 84
14.13 Každá věc se dá udělat více způsoby … . . . 84

15 Derivace a modely založené na derivaci **87**
15.1 Úvod . . . 87
15.2 Numerická simulace v rovnici ochlazování . . . 88
15.3 Simulace pro více počátečních podmínek . . . 91
15.4 Simulace pro model se zpožděním . . . 91

16 Diferenciální rovnice 1 **95**
16.1 Část A: Řešení diferenciální rovnice (modifikace existujícího kódu) . . . 95
16.2 Část B: Řešení diferenciální rovnice (podrobnější pohled pod kapotu) . . . 97
16.3 Část C: Model rovnováhy počtu druhů na ostrovech . . . 101

17 Diferenciální rovnice 2 **105**
17.1 Exponenciální růst . . . 105
17.2 Exponenciální růst k vodorovné asymptotě . . . 106
17.3 Logistický růst . . . 106

18 Diferenciální rovnice 3 **109**
18.1 Lov v logistické rovnici . . . 109
18.2 Alleeho efekt . . . 112
18.3 Populace pod predačním tlakem . . . 113

19 LaTeX **115**

20 Maticové modely **117**
20.1 Leslieho model . . . 117
20.2 Model jelena evropského (red deer, *Cervus elaphus*) . . . 120
20.3 Tropický strom . . . 121
20.4 Kareta obecná (Loggerhead sea turtle) . . . 122

21 Levinsův model metapopulací **125**
21.1 Populace ve fragmentovaném prostředí . . . 125
21.2 Matematický model . . . 125
21.3 Důsledky modelu . . . 126
21.4 Možné úkoly . . . 126

22 Modely konkurence populací **127**
22.1 Lotkův–Volterrův model konkurence dvou populací . . . 127
22.2 Konkurence tří populací . . . 127

23 Modely dravce a kořisti **129**
23.1 Lotkův a Volterrův mode dravce a kořisti . . . 129
23.2 Model dravce a kořisti s Hollingovou trofickou funkcí . . . 129
23.3 Model dravce a dvou populací kořisti . . . 129
23.4 Model dravce a kořisti s kanibalismem dravce . . . 129

24 Práce na zápočtovém projektu **131**

III Python konstrukce a ukázky **133**

25 Základní konstrukce **135**
25.1 Sestavení seznamu . . . 135
25.2 Cyklus . . . 135
25.3 Větvení . . . 136

26 Cykly **137**
26.1 Prostá iterace . . . 137
26.2 Iterace přes seznam . . . 137
26.3 Iterace přes více seznamů současně . . . 137
26.4 Iterace se sledováním pozice v seznamu . . . 138

26.5 Praktická poznámka . . . 138
26.6 Předčasné ukončení . . . 138
26.7 Praktická ukázka . . . 138

27 Kreslení funkcí jednoduše **141**

28 Kreslení funkcí pro pokročilé **143**

29 Diferenciální rovnice pro začátečníky **145**

30 Diferenciální rovnice pro pokročilejší **147**

31 Tabulky **149**
31.1 Tabulky z externího souboru . . . 149
31.2 Grafy . . . 150
31.3 Tvorba tabulek . . . 151

32 Obrázky, grafy **153**

33 Řešení diferenciální rovnice **155**
33.1 Varianta 1: Chceme funkční předpis . . . 155
33.2 Varianta 2: Chceme data pro předem známé časy . . . 156
33.3 Různé hodnoty parametrů . . . 156

34 Tabulky s None **159**

35 Tabulky s víceúrovňovými nadpisy sloupců **163**

IV Literatura a rejstřík **167**

36 Literatura **169**

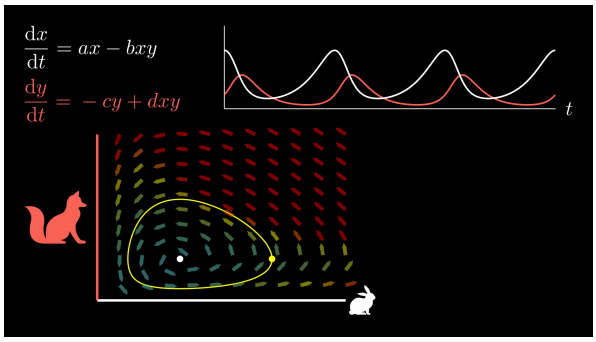
37 Rejstřík **171**

Literatura **173**

Proof Index **175**

PDF verze

```
Poslední aktualizace:  24. 3. 2023␣
↪11:47:53
```



Obr. 1: Lotkův-Volterrův model dvoudruhové populace dravce a kořisti byl v první polovině dvacátého století jedním z prvních počinů při snaze využít matematické modelování k objasnění přírodních fenoménů spojených s velikostmi populací živočichů. Model vysvětlil vznik pozorovaných oscilací ve společenstvu dravce a kořisti a také poněkud překvapivou skutečnost, že omezení lovu způsobí posun rovnováhy ve prospěch dravců.

O předmětu

Předmět je přednášen na oboru Myslivost. Obsahuje matematický aparát používaný v matematické biologii k popisu vývoje populací, jejich vzájemné interakce a schopnosti přežívat. Webová stránka předmětu s popisem požadavků na ukončení je na webu vyučujícího.

Budeme používat matematické postupy a zkoumat matematické modely, ale předmět není matematikou. Nemá tedy ambice všechny pojmy definovat přesným způsobem v matematice obvyklým. Přesné definice těchto pojmů je snadné najít v literatuře, například [5].

Motivace 1, záchrana Lišky ostrovní

Na úvod jako motivaci uveďme jeden optimistický příběh.

Liška ostrovní (*Urocyon littoralis*) je jedinečný živočišný druh, endemit žijící jenom na ostrůvcích okolo Kalifornie. Velká jako kočka, důvěřivá (díky absenci přirozených nepřátel), jako druh citlivá a zranitelná (malá genetická variabilita, malá odolnost vůči nemocem zavlečeným z pevniny). Jedna z nejmenších psovitých šelem. Umí šplhat po stromech. Vlivem činnosti člověka se dostala do velkých potíží. Na ostrově San Miguel klesla populace z 450 dospělých jedinců v roce 1994 na 15 v roce 1999 (odkaz). Podobná situace byla i na ostatních

Obr. 2: *Liška ostrovní, poddruh žijící na ostrově Santa Cruz, foto: https://wikimedia.org*

ostrovech, z nichž každý je osídlen samostatným poddruhem lišky ostrovní. Příčinou úhynu by celý řetězec událostí: produkce insekticidu DDT, farmaření a rozmnožení zdivočelých prasat, vytlačení orla bělohlavého (*Haliaeetus leucocephalus*) orlem skalním (*Aquila chrysaetos*). Dříve vrcholný predátor na ostrově se stal najednou kořistí a byl těsně před vyhubením. Naštěstí se podařilo situaci pro lišku zachránit, zajistit podmínky ve kterých je populace stabilní a populaci lišek opětovně rozmnožit. Nyní je liška ostrovní „pouze“ téměř ohrožená. Jedná se o jeden z nejúspěšnějších záchranných programů pro savce. Komplexní program zahrnoval vybití divokých prasat, přesídlení orlů skalních, návrat orlů bělohlavých, umělé rozmnožení lišek, jejich návrat do přírody a jejich vakcinaci proti zavlečeným chorobám. To vše za jednu dekádu.

- Článek na idnes.cz z roku 2000 o hrozícím vyhubení.
- Youtube video z roku 2018 o záchraně ostrovních lišek.
- Matematický model interakce mezi liškami, orly, skunkem a zdivočelými prasaty.
- Řešení modelu v prostředí Jupyter notebooku

  Specifika ostrovní biogeografie si představíme ve *třetí přednášce*. Modelům interakce živočišných druhů ve vztahu *konkurence* nebo *predace* se budeme věnovat koncem semestru. Jednoduchý Jupyter zápisník, ukazující, že umíte zkombinovat text s výpočty a umíte spustit model a nějakým způsobem vizualizovat řešení bude obsahem seminární práce, nutné pro ukončení předmětu.

Motivace 2, záchrana populace želv

Mořské želvy obecně mají poměrně komplikovaný život-

Obr. 3: *Kareta obecná, foto: https://wikimedia.org*

ní cyklus. Vylíhnutí z vajíček probíhá na souši, další vývoj v moři. Ochrana se tradičně soustřeďovala na ochranu pláží pro kladení vajíček a umožnění vylíhnutým želvičkám dostat se k vodě. To je efektní a snadno proveditelná činnost. Je to ale i efektivní? Je to opravdu účinná strategie pro podporu populace želv?

Matematický rozbor přínosu jednotlivých životních fází k celkovému růstu populace ukázal, že pro udržení druhu ochrana vajíček a čerstvě vylíhnutých mláďat nestačí. Ukázal, že tato činnost má malý vliv. Želva se o svá mláďata nestará a jenom malý zlomek se dožije dospělosti. Ukázalo se, že pro podporu populace je důležité chránit dospělé jedince, na kterých stojí břímě rozmnožování. Je nutné snížit riziko, že tyto jedince z populace odstraníme například jako vedlejší efekt rybolovu. Protože se ukazuje, že tato aktivita je pro populaci podstatná (dospělí jedinci mají velkou takzvanou *reprodukční hodnotu*), má smysl hledat cesty a řešení. V případě rybářských sítí je to například zařízení zvané TED, Turtle Excluder Device, které umožní dospělým želvám (a velkým rybám), na nichž závisí přežití druhu, uniknout z rybářských sítí. Protože se jedná se o poměrně jednoduché a levné zařízení, je poměr cena/užitek obrovský.

> Model populace s různými životními etapami si představíme na *šesté přednášce*. Na *sedmé přednášce* si ukážeme nástroj na analýzu vlivu parametrů na celkový růst populace a v *jednom ze cvičení* se budeme blíže věnovat zmíněnému modelu populace karety.

Co budeme dělat

V předmětu si představíme základní metody modelování jevů a efektů známých z ekologie a populační biologie. Z těchto oblastí máme celou řadu poznatků.

- Populace nerostou neomezeně, ale jenom do nosné kapacity prostředí (Verhulst, 1838).
- Větší územní celky mají pestřejší druhové složení. (Mac Arthur, Wilson, 1967).
- Malé populace mohou mít malou nebo zápornou rychlost růstu per capita (Allee, cca 1930–1940).
- Věkově strukturovaná společenstva konvergují ke stabilní věkové skladbě (Leslie, 1945).
- Při konkurenci dvou a více druhů dojde za určitých podmínek ke konkurenčnímu vyloučení některých druhů (Gause, 1934).
- Ve fragmentovaném prostředí je nutno chránit i migrační trasy mezi fragmenty a také fragmenty, kde se daná populace nenachází (Levins, 1969).
- V přírodě dochází k oscilacím, například ve společenstvech dravce a kořisti (Lotka, Volterra, 1920), u dlouhověkých cikád, při některých chemických reakcích (Bělousov, Žabotinski, 1951), při periodickém přemnožování obaleče v kanadských lesích (Ludwig, Jones, Holling, 1978).
- Organismy produkují enzymy a bílkoviny a jejich fungování je závislé na rychlosti, s jakou dokáží syntetizovat potřebný enzym a na tom, jak dokáží udržet jeho stabilní hladinu. Výhodnější než jednoduchá konstantní produkce enzymu je, pokud si organismus vytvoří evolucí taktiku směřující ke zrychlení syntézy a zvýšení robustnosti stacionárního stavu, tj. zvýšení odolnosti proti narušení rovnováhy (viz [2]).

Ukážeme si, že uvedené efekty nejsou ničím jiným, než nevyhnutelným důsledkem principů, podle kterých funguje dané společenství. Matematickým modelováním převedeme představy o vztazích v populacích na pozorovatelná data. Zpravidla budeme sledovat velikost populace a její vývoj v čase.

Pokud matematický model vykazuje shodu s pozorovanou skutečností, máme všechny důvody domnívat se, že naše představy o principech fungování společenství jsou správné a to nám umožní sledovat, jak se situace bude měnit při změně parametrů jako jsou změna prostředí nebo lov či sběr členů této populace. Matematika umožní levné, snadné, etické a bezpečné experimenty s danou populací.

O metodách výuky

Cvičení jsou z výpočetního hlediska zaměřeny na využití jazyka Python. To je moderní a perspektivní skriptovací jazyk, vhodný pro manipulaci s daty a přístupný i neprogramátorům.

- Díky Pythonu zpřístupníme možnosti řešení matematických modelů i nematematikům. Vyhneme se obtížným výpočtům, kvůli kterým bychom se museli naučit používat netriviální matematický aparát, jako jsou derivace a integrály.

- Nebojte se, nebudete programovat. Měli byste pasivně rozumět kódu z přednášek a cvičení a umět jej přizpůsobit jinému modelu nebo jiné situaci.
- Příklady pro **cvičení** jsou zápisníky prostředí Jupyter, ve kterém Python budeme spouštět. Je možné otevřít v online prostředí nebo na svém lokálním PC. Online možnosti jsou přístupné přes horní lištu pod ikonkou rakety.
  - JupyterHub je volba, kterou budeme používat během cvičení. Touto volbou se vám naklonují materiály do vlastního adresáře a budete moci s touto jejich kopií pracovat. Co se přesně stane a jaké jsou možnosti je popsáno v dokumentaci k nbgitpuller. Budete pracovat na serveru univerzity, abychom měli všichni stejné prostředí. Zde se také bude pracovat na zápočtovém projektu. Účet na jupyter.mendelu.cz mají studenti předmětu. Login je stejný jako do UIS (například `xnovak65`), přednastavené heslo máte v UIS, návod jak heslo najít je na přihlašovací stránce JupyterHubu.
  - Binder založí dočasně server na službě mybinder.org a umožní Vám experimentovat s kódem, spouštět příkazy. Práce je dočasná, pokud po ukončení nechcete o práci přijít, musíte si zápisník stáhnout na lokální PC. Je možné doinstalovat balíčky, například knihovnu intersect příkazem `! pip install intersect` (vykřičník na začátku informuje o tom, že se jedná o příkaz operačního systému a ne příkaz jazyka Python).
  - Colab vám po přihlášení ke Google účtu uloží zápisník na Google disk. Prostředí je trochu jiné, ale práce je zpravidla svižnější na odezvu. Zápisník se ukládá a je možné sledovat změny a ukládat verze. Výsledek práce je možno uložit na GitHub do svého repozitáře nebo jako Gist. Je možné doinstalovat chybějící balíčky stejně jako v předchozím bodě pro Binder.
  - Live Code umožní spouštět příklady přímo na stránce, kterou čtete. Volba je nejméně svižná, ale použitelná.
  - Offline práce je možná, pokud máte nainstalovaný Python, Jupyter a s tím spojený ekosystém. Doporučenou volbou pro nejběžnější operační systémy (Linux, Windows) je Anaconda a jako editor buď Jupyter zápisník v prohlížeči nebo editor Visual Studio Code a pluginy Python, Pylance, Python for VSCode, Jupyter.
- **Přednášky** jsou psány jazykem MyST a je možné je otevřít v online prostředí, které tento jazyk podporuje. To je v zimě 2022 částečně Binder a JupyterHub z online služeb a Jupyter Notebook jako offline volba. (Nejsou renderovány definice, věty a některá další prostředí, ty se zobrazují jako programový kód.) Vývoj však jde rychle dopředu…

Varování: Pokud pracujete na lokálním počítači, máte data pod kontrolou a nikdo jiný je nevidí. Při práci na JupyterHubu na Mendelově univerzitě data může vidět i admin serveru. (Toho budeme využívat při práci na zápočtovém projektu.) Při práci na serverech jako Colab či Binder jdou data přes servery dalších vlastníků a nemáte data plně pod kontrolou. Proto není vhodné pracovat s osobními daty a citlivým obsahem.

Troubleshooting

V příkazech je chyba a je těžké ji najít

Příkazy pište po malých krůčcích a teprve poté vše spojte. Pokud vidíte dlouhou chybovou hlášku, je zpravidla nejinformativnější její konec. Pokud nevíte o co jde, můžete část chybového hlášení nakopírovat do vyhledávacího políčka google a pátrat po radách k uvedenému problému.

Učební text se zasekává při otevírání na jupyter.mendelu.cz

Pokud si naklonujete učební text k sobě a něco v něm upravíte, může při dalších verzích dojít ke kolizi a nestáhne se vám aktuální verze. Nejrychlejší řešení je přepnout se do konzole (v seznamu souborů použít New, Terminal) a tam zadat příkazy OS Linux. Můžete použít `mv dmp dmp_kopie` pro přejmenování. Zůstanou Vám i upravené soubory v adresáři dm_kopie. Pokud svou práci ukládáte jinam (doporučeno do hlavního adresáře nebo do svého podadresáře), můžete adresář smazat příkazem `yes | rm -r dmp` a poté se při otevření souboru z učebního textu naklonuje celá nová verze.

Část I

Přednášky

KAPITOLA 1

Funkce a vlastnosti funkcí

Co se dozvíte v tomto textu

Při studiu libovolného fenoménu dříve či později skončíme u snahy kvantifikovat velikosti znaků, které nás zajímají a sledovat vztahy mezi jednotlivými znaky. Jinými slovy, sledujeme vybrané veličiny a pomocí funkcí vyjadřujeme vztahy mezi těmito veličinami navzájem, nebo mezi těmito veličinami a parametry prostředí. Typickým případem je sledování velikosti populace a jejího vývoje v čase. Tím je možné odhalit některé děje, jako například periodické kolísání populace rysa a zajíce v okolí Hudsonova zálivu.

V úvodním textu si zopakujeme pojmy potřebné pro práci s funkcemi a v některých částech si pojmy rozšíříme nad úroveň střední školy.

Foto: Rys kanadský (Lynx canadensis). Pravidelné fluktuace populací rysa a zajíce slouží v literatuře k vysvětlení dynamiky společenstva dravce a kořisti. Autor Keith Williams, https://commons.wikimedia.org

Při studiu přírodních dějů a jevů sledujeme vybrané charakteristiky, kterým je možno přiřadit číselné hodnoty spojené s jednotkou. Například velikost populace v kusech jedinců. Takové charakteristiky se nazývají veličiny.

Pokud u dané populace sledujeme více veličin, mohou mezi těmito veličinami být jisté vztahy. Takové vztahy vyjadřujeme pomocí funkcí. Jednou z veličin může být třeba čas v letech a druhou z veličin velikost populace. Funkce potom udává souvislost mezi velikostí populace a časem. Může například jít o vzorec, do kterého dosadíme čas a vzorec nám ukáže velikost populace.

1.1 Funkce jedné proměnné

Formální definice funkce je v následujícím odstavci. Zpravidla nebývá pro pochopení práce s funkcemi nutná a postačí intuitivní chápání tohoto pojmu jako vztahu mezi dvěma veličinami nebo mezi dvěma proměnnými.

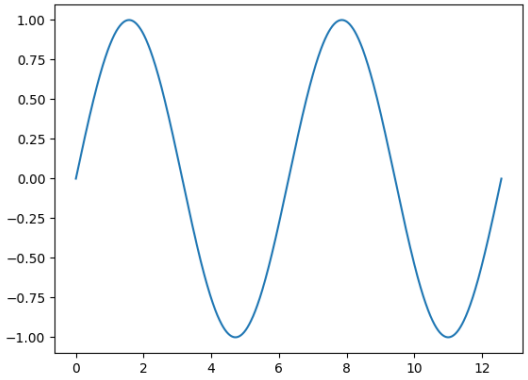
Definice (Funkce)

Buďte A a B neprázdné podmnožiny množiny reálných čísel. Pravidlo f, které každému prvku množiny A přiřadí jediný prvek množiny B se nazývá *funkce* (přesněji: *reálná funkce jedné reálné proměnné*). Zapisujeme $f : A \to B$. Skutečnost, že prvku $a \in A$ je přiřazen prvek $b \in B$ zapisujeme $f(a) = b$. Přitom říkáme, že b je *obrazem prvku* a při zobrazení f, resp. že a je *vzorem prvku* b při zobrazení f.

Hodnoty x, pro které je jejich obraz roven nule, se nazývají *kořeny* funkce.

Zjednodušeně řečeno, funkce je pravidlo, které zadanému číslu na vstupu dokáže přiřadit pevně definované číslo na výstupu.

Funkce vizualizujeme pomocí grafu v kartézské soustavě souřadnic. Na vodorovnou osu vynášíme vzory, na svislou obrazy. V prakticky zajímavých a využívaných případech zpravidla body grafu vyplní křivku v rovině.



1.2 Vlastnosti funkcí

Pro práci s funkcemi si všímáme vlastností, které jsou určitým způsobem vypovídající o tom, jak se funkce chová na definičním oboru nebo na jeho podmnožině. Zajímá nás zejména

- jestli při seřazení vstupních dat podle velikosti jsou výstupy seřazeny stejným způsobem (rostoucí funkce) nebo naopak (klesající funkce),
- jestli výsledek na výstupu funkce jednoznačně determinuje data na vstupu (prostá funkce),
- jak určit data na vstupu z výsledku na výstupu, pokud tato data určit jdou (inverzní funkce),
- jestli data na výstupu funkce nutně zůstávají mezi nějakými pevnými hodnotami (ohraničenost)
- a jestli se funkce pro hodně velká data chovají přibližně stejně jako nějaké jednodušší funkce (asymptotika).

To bylo neformální představení základních vlastností, kterých si všímáme u funkcí. Taková představa většinou stačí. Přesné definice jsou následující.

Definice (Monotonie funkce)

Nechť f je funkce a $M \subseteq \mathrm{Dom}(f)$ podmnožina definičního oboru funkce f

- Řekneme, že funkce f je na množině M *rostoucí* jestliže pro každé $x_1, x_2 \in M$ s vlastností $x_1 < x_2$, platí $f(x_1) < f(x_2)$.
- Řekneme, že funkce f je na množině M *klesající* jestliže pro každé $x_1, x_2 \in M$ s vlastností $x_1 < x_2$, platí $f(x_1) > f(x_2)$.
- Řekneme, že funkce f je na množině M *(ryze) monotonní* je-li buď rostoucí, nebo klesající na M.

Nespecifikujeme-li množinu M, máme na mysli, že uvedená vlastnost platí na celém definičním oboru funkce f.

Následující definice zavádí třídu funkcí, u kterých se z výsledku dají zrekonstruovat vstupní data.

Definice (Prostá funkce)

Nechť f je funkce a $M \subseteq \mathrm{Dom}(f)$ podmnožina definičního oboru funkce f. Řekneme, že funkce f je *prostá*, jestliže každý obraz má jen jediný vzor, tj. pro každé $y_0 \in f(M)$ existuje jediné $x \in M$ s vlastností $f(x) = y_0$. Nespecifikujeme-li množinu M, máme na mysli, že uvedená vlastnost platí na celém definičním oboru funkce f.

Následující věta vyjadřuje, že pokud je funkce prostá, je možno ji odebrat z obou stran rovnice a výsledkem je rovnice ekvivalentní.

Věta (Rovnice s prostou funkcí)

Pokud je f prostá funkce a platí

$$f(x) = f(a),$$

potom platí $x = a$.

Pokud je funkce prostá, je možné z výsledku zrekonstruovat vstupní data. Následující definice dává název funkci, která toto přiřazení vstupních dat k výsledkům realizuje.

Definice (Inverzní funkce)

Nechť funkce $f : A \to B$ je prostá. Pravidlo, které každému x z množiny $f(A)$ přiřadí to (jediné) y, pro které platí $f(y) = x$ se nazývá *inverzní funkce* k funkci f, označujeme f^{-1}.

Například k druhé odmocnině je inverzní funkcí druhá mocnina. K násobení dvojkou je inverzní funkcí dělení dvojkou.

Definice (Ohraničená funkce)

Funkce se nazývá *shora ohraničená na množině* M, pokud existuje reálné číslo K takové, že platí

$$f(x) < K$$

pro všechna x z množiny M.

Analogicky pomocí opačné nerovnosti definujeme ohraničenost na množině M zdola.

Funkce je ohraničená na množině M, pokud je ohraničená současně shora i zdola.

Pokud množinu M vynecháme, máme na mysli ohraničenost na celém maximálním definičním oboru.

Příklad 1.2.1

Sinus a kosinus jsou ohraničené funkce. Exponenciální funkce je zdola ohraničená a není shora ohraničená. Na intervalu konečné délky je ohraničená i exponenciální funkce.

Definice (Landauova notace)

Nechť g je funkce kladná pro velká x. Řekneme, že funkce f je třídy $\mathcal{O}(g)$, píšeme

$$f(x) = \mathcal{O}(g(x)),$$

pokud je funkce $|f(x)|/g(x)$ ohraničená pro velká x, tj. pokud pro nějakou konstantu M platí

$$|f(x)| \leq Mg(x)$$

pro velká x.

Řekneme, že funkce f je třídy $O(g)$ v okolí bodu a, pokud uvedené vztahy platí na nějakém intervalu obsahujícím ve svém vnitřku bod a.

Je-li funkce $f(x) = O(x^2)$, existuje konstanta K s vlastností

$$|f(x)| \leq Kx^2$$

pro velká x.

1.3 Elementární funkce

Shrneme si nejdůležitější pojmy spojené s elementárními funkcemi. Jedná se o znalosti ze střední školy a možná jejich mírné rozšíření. Proto není potřeba vždy tyto funkce na současném místě formálně přesně definovat.

1.3.1 Přímá a nepřímá úměrnost

Je to až k nevíře, ale k popisu obrovského množství dějů stačí čtyři základní operace: sčítání, odčítání, násobení a dělení. Vzhledem k požadavku na konzistenci fyzikálních jednotek se nejčastěji setkáváme s násobením a dělením a proto funkce pracující s těmito operacemi mají výsadní postavení. Dokonce si vysloužily pojmenování běžně užívané i mezi nematematiky: přímá a nepřímá úměrnost. Je to formální popis situace, kdy souvislost mezi dvěma veličinami je zprostředkována násobením konstantou (přímá úměrnost), nebo kdy je násobením konstantou zprostředkována souvislost mezi jednou veličinou a převrácenou hodnotou druhé veličiny (nepřímá úměrnost).

Definice (Přímá a nepřímá úměrnost)

Veličina y je *přímo úměrná* veličině x jestliže existuje konstanta k taková, že platí

$$y = kx.$$

Veličina y je *nepřímo úměrná* veličině x jestliže existuje konstanta k taková, že platí

$$y = \frac{k}{x}.$$

Je-li veličina y úměrná veličině x, píšeme $y \sim x$ nebo $y \propto x$. Je-li navíc konstanta úměrnosti blízká jedničce, tj. x a y jsou blízké, píšeme $y \approx x$.

Pro nepřímou úměrnost píšeme podobně $y \sim \frac{1}{x}$, $y \propto \frac{1}{x}$ a $y \approx \frac{1}{x}$ s využitím toho, že nepřímá úměrnost mezi dvěma veličinami je vlastně přímá úměrnost mezi jednou veličinou a převrácenou hodnotou veličiny druhé.

Grafem přímé úměrnosti je přímka procházející počátkem. Přímá úměrnost může být i klesající, je-li konstanta k záporná.

1.3.2 Mocninné funkce

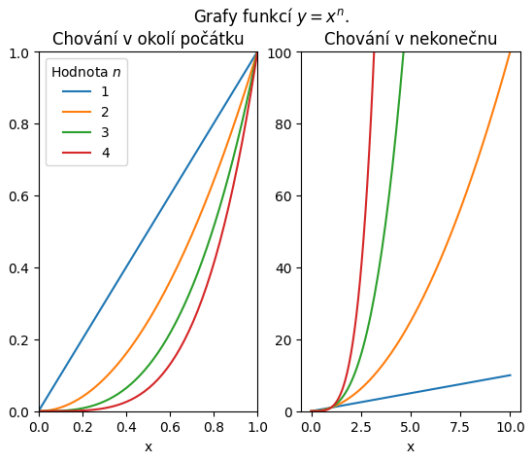
Mocninná funkce je funkce ve tvaru

$$y = x^k,$$

kde x je proměnná (základ mocniny) a k číselný parametr (exponent). Exponent nemusí být celé číslo.

Chování mocninných funkcí je určeno exponentem. Pro kladné exponenty vyšší exponent znamená rychlejší růst pro velké hodnoty argumentu a rychlejší pokles k nule v okolí počátku.

Nakreslíme grafy mocninných funkcí. Pro názornost je vykreslíme dvakrát vedle sebe a jednou omezíme grafy na okolí počátku. Níže si můžete rozkliknout skript pto výpočet funkčních hodnot a vykreslení obrázku.



1.3.3 Polynomy

Polynom je součet násobků mocninných funkcí s celočíselným exponentem. Stupněm polynomu rozumíme hodnotu největšího exponentu. Například

$$f(x) = x^3 - 2x^2 + 1$$

je polynom třetího stupně. Polynomy druhého stupně se nazývají kvadratické funkce, polynomy třetího stupně se nazývají kubické funkce.

Chování polynomu v nekonečnu určuje člen s nejvyšší mocninou. Například polynom

$$y = -2x^4 + 3x^3 + 5x + 2$$

je pro velká x záporný, stejně jako funkce $y = -2x^4$. Pro x jdoucí k nekonečnu jdou funkční hodnoty do mínus nekonečna.

Kvadratické funkce

Kvadratickou funkcí rozumíme funkci ve tvaru polynumu druhého stupně

$$y = ax^2 + bx + c,$$

kde $a \neq 0$. Je-li výraz $b^2 - 4ac$ kladný, má polynom dva reálné kořeny

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

a platí

$$y = a(x - x_1)(x - x_2).,$$

Grafem kvadratické funkce je parabola s průsečíky na ose v bodech $x = x_1$ a $x = x_2$. Pro $a > 0$ je otočená vrcholem dolů, v opačném případě vrcholem nahoru.

Vrchol paraboly, která má dva reálné kořeny, má x-ovou souřadnici uprostřed mezi kořeny, tj.

$$x_{\text{vrchol}} = \frac{x_1 + x_2}{2} = -\frac{b}{2a}.$$

Vzorec

$$x_{\text{vrchol}} = -\frac{b}{2a}.$$

platí i když parabola nemá reálné nulové body a je celá nad vodorovnou nebo pod vodorovnou osou.

Logistický růst

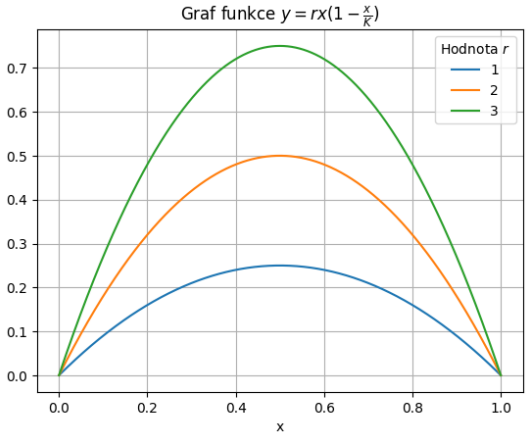
Kvadratická funkce se v modelech populací vyskytuje často v modelech zohledňujících vnitrodruhovou konkurenci. V takovém případě zpravidla předpokládáme, že rychlost růstu populace je úměrná velikosti populace a volnému místu v prostředí s konečnou nosnou kapacitou. Toto volné místo zpravidla vyjadřujeme procentem obsazenosti prostředí. Jednoduchým matematickým modelem rychlosti takového růstu je funkce

$$f(x) = rx\left(1 - \frac{x}{K}\right),$$

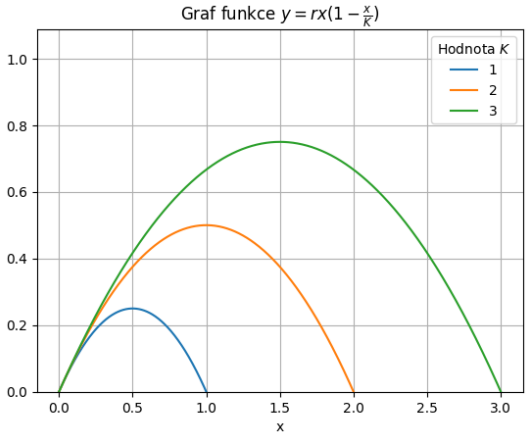
kde K je *nosná kapacita prostředí* a r je *invazní parametr* (specifická rychlost růstu pro malé velikosti populace). Tato kvadratická funkce proměnné x s parametry r a K má graf parabolu otočenou vrcholem nahoru s nulovými body $x = 0$ a $x = K$.

Pro rostoucí r a pevné K se nemění poloha nulových bodů funkce, ale rostou její funkční hodnoty.

```
             x          1          2          3
0     0.000000   0.000000   0.000000   0.000000
1     0.001001   0.001000   0.002000   0.003000
2     0.002002   0.001998   0.003996   0.005994
3     0.003003   0.002994   0.005988   0.008982
4     0.004004   0.003988   0.007976   0.011964
..         ...        ...        ...        ...
995   0.995996   0.003988   0.007976   0.011964
996   0.996997   0.002994   0.005988   0.008982
997   0.997998   0.001998   0.003996   0.005994
998   0.998999   0.001000   0.002000   0.003000
999   1.000000   0.000000   0.000000   0.000000

[1000 rows x 4 columns]
```



Pro rostoucí K a pevné r se nemění směr růstu paraboly v okolí počátku, ale roste se kladný kořen a rostou i funkční hodnoty.



1.3.4 Racionální funkce

Racionální funkce jsou podílem dvou polynomů.

- Pokud je stupeň polynomu v čitateli menší než stupeň polynomu ve jmenovateli, funkce se v nekonečnu blíží k nule.
- Pokud mají čitatel i jmenovatel stejný stupeň, funkce má v nekonečnu vodorovnou asymptotu různou od nuly.

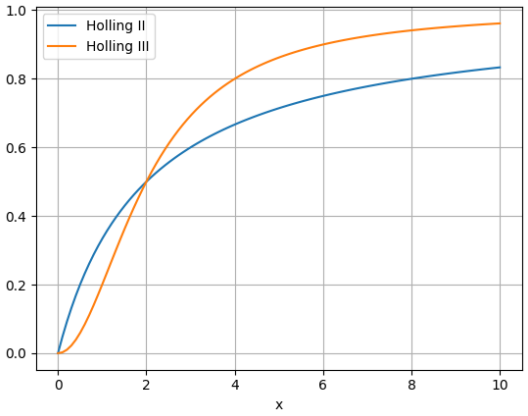
V ekologii se často používají funkce, modelující specifickou funckionální odezvu. Setkáme se s nimi později v modelech dravce a kořisti [6], [3]. Významnými funkcemi jsou takzvané funkce Hollingova typu, z nichž zmíníme typ Holling II a Holling III. V obou případech se jedná o kladné rostoucí funkce vycházející z počátku souřadné soustavy a konvergující k vodorovné asymptotě, přičemž Holling II vychází z počátku souřadnic pod ostrým úhlem a Holling III s vodorovnou tečnou a má tedy pomalejší náběh. Matematickými modely ve formě racionálních funkcí pro odezvu tohoto typu mohou být funkce

$$f(x) = \frac{x}{x + a}$$

pro Holling II a

$$f(x) = \frac{x^k}{x^k + a^k}, \quad k > 1$$

pro Holling III. (V případě vzorce pro typ Holling III číslo k nemusí být celé a formálně se tedy nebude jednat o podíl dvou polynomů.)



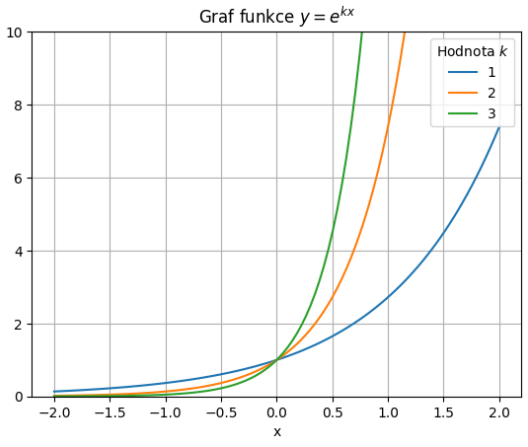
1.3.5 Exponenciální funkce a logaritmus

Exponenciální funkce je mocnina s pevným základem a proměnným exponentem. Exponenciální funkce je prostá a její inverze je logaritmická funkce. V modelech nejčastěji pracujeme s přirozenými logaritmy a exponenciální funkcí o základu e. Vzhledem k rovnosti

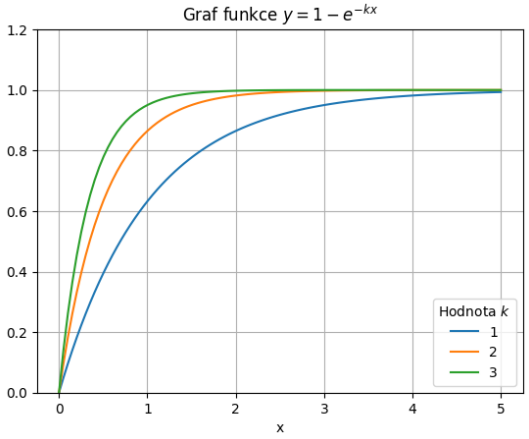
$$a^x = e^{x \ln a}$$

základ exponenciální funkce není podstatný a libovolnou exponenciální funkci je možné přepsat na exponenciální funkci o libovolném základě.

Funkce $y = e^{kx}$ pro k kladné je rostoucí v nekonečnu (roste rychleji než polynom libovolného stupně) a klesající v mínus nekonečnu (klesá k nule rychleji než libovolná racionální funkce). Hodnota k reguluje rychlost růstu do nekonečna a poklesu k vodorovné asymptotě.



Funkce $y = a + e^{-kx}$ (kde k je kladné) klesá shora k vodorovné asymptotě $y = a$ tím rychleji, čím je vyšší hodnota k. Funkce $y = a - e^{-kx}$ se chová analogicky, ale k asymptotě roste.



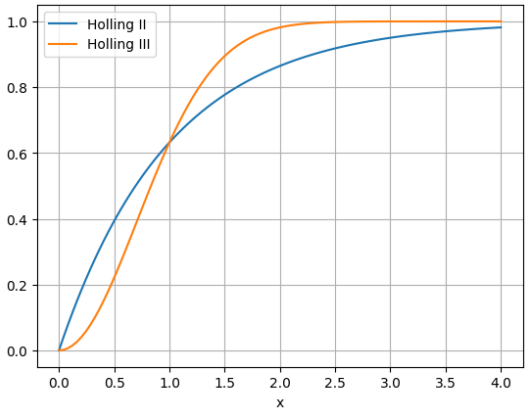
Pomocí exponenciálních funkcí je možno modelovat funkce Hollingova typu II a III například vztahy

$$y = 1 - e^{-kx}$$

pro typ Holling II a

$$y = 1 - e^{-kx^2}$$

pro typ Holling III.

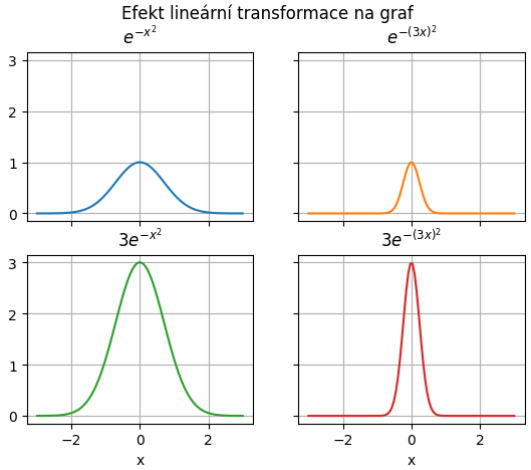


Tyto funkce mají podobný tvar jako analogické funkce uvedené v kapitole o racionálních funkcích, ale charakter závislosti je poněkud jiný a na základě empirických dat se zpravidla rozhodujeme, který způsob popisu nejlépe vyhovuje zkoumané realitě.

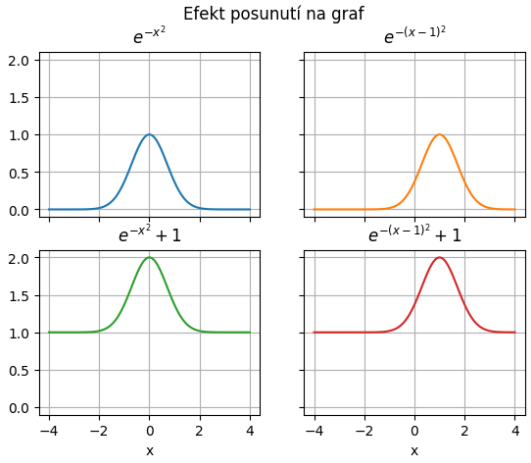
Exponenciální funkce s kladným koeficientem v exponentu modeluje růst, kdy rychlost růstu veličiny je úměrná velikosti této veličiny, tj. s tím jak veličina roste se zrychluje i rychlost jejího růstu. Exponenciální funkce se záporným koeficientem v exponentu se vyskytuje často tam kde systém konverguje do stacionárního stavu. Potom často vzdálenost od stacionárního stavu exponenciálně klesá s časem.

1.3.6 Složené funkce

Často je nutné modifikovat chování funkce podle aktuální potřeby. Například zařídit, aby funkce nerostla k jedničce, ale k jiné hodnotě. Někdy potřebných požadavků můžeme dosáhnout tak, že namísto s funkcí pracujeme s jejím násobkem, nebo do ní namísto proměnné dosazujeme násobek proměnné. Tedy namísto s funkcí $y = f(x)$ pracujeme s funkcí $y = k_1 f(k_2 x)$, jejíž graf dostaneme z grafu funkce $y = f(x)$ natažením k_1-krát ve svislém směru a zúžením k_2-krát ve vodorovném směru. Viz následující obrázek.



Podobně je možno sledovat vliv posunutí. Přičtení konstantní hodnoty k funkci značí posun grafu ve vertikálním směru, přičtení konstantní hodnoty k argumentu posun ve vodorovném směru. Přičtení kladné hodnoty způsobí posun grafu nahoru resp. doleva, přičtení záporné hodnoty posun na opačnou stranu.



1.4 Funkce více proměnných a vektorové funkce

V praxi se neomezujeme na funkce, které mají na vstupu jedno číslo a vracejí jako funkční hodnotu opět jedno číslo, ale proměnných na vstupu i na výstupu může být více.

1.4.1 Skalární funkce více proměnných

Typickým případem kdy na vstupu funkce je více veličin a na výstupu veličina jediná je rychlost růstu populace, jejíž nosná kapacita je ovlivněna jinou populací. Jsou-li velikost populace x a velikost populace konkurenta y, můžeme k modelování takové rychlosti růstu použít funkce

$$f(x,y) = rx\left(1 - \frac{x}{K + \alpha y}\right)$$

nebo

$$f(x,y) = rx\left(1 - \frac{x}{K} - \alpha y\right),$$

kde nám kromě invazního parametru r a nosné kapacity prostředí K přibyl parametr α charakterizující sílu mezidruhové konkurence. Je-li f funkce dvou proměnných, píšeme $f\colon \mathbb{R}^2 \to \mathbb{R}$.

Vizualizace funkce dvou proměnných může být realizována jako plocha ve 3D prostoru, což však někdy nedává zcela názornou představu. Další možností je s druhou proměnnou zacházet jako s parametrem a kreslit graf funkce jedné proměnné pro vybrané pevné hodnoty druhé proměnné.

1.4.2 Vektorová funkce jedné proměnné

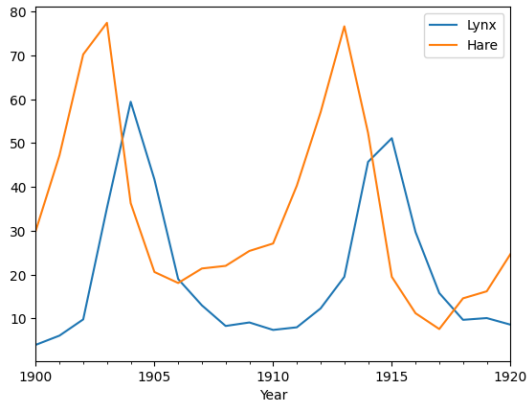
Vektorová funkce jedné proměnné je funkce, která má na vstupu jednu proměnnou a na výstupu více proměnných, které zpravidla uvažujeme jako komponenty vektoru. Typickým případem je časový průběh velikosti dvou populací.

Vizualizovat funkci můžeme tak, že nakreslíme graf každé komponenty samostatně (pokud kreslíme do jednoho obrázku, můžeme použít dvě osy y). Je-li výstupem vektor v rovině, můžeme funkci vizualizovat tak, že kreslíme uspořádané dvojice pro různé hodnoty vstupních dat.

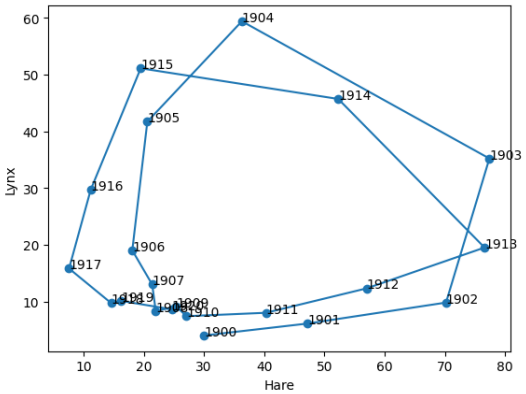
Na obrázku je vektorová funkce, kde vstupem je rok a výstupem je velikost populace zajíce běláka (angl. hare) a sněžného rysa (angl. lynx) v oblasti Hudsonova zálivu. V matematické biologii je to jeden z nejznámějších modelů vzájemného působení populací. Data stáhneme ve formě csv souboru do tabulky a pro kontrolu vytiskneme začátek tabulky.

```
        Year  Lynx  Hare
0 1900-01-01   4.0  30.0
1 1901-01-01   6.1  47.2
2 1902-01-01   9.8  70.2
3 1903-01-01  35.2  77.4
4 1904-01-01  59.4  36.3
```

Vykreslení dat pro jednotlivé druhy je snadné, stačí zadat, jaká veličina má být na vodorovné ose.



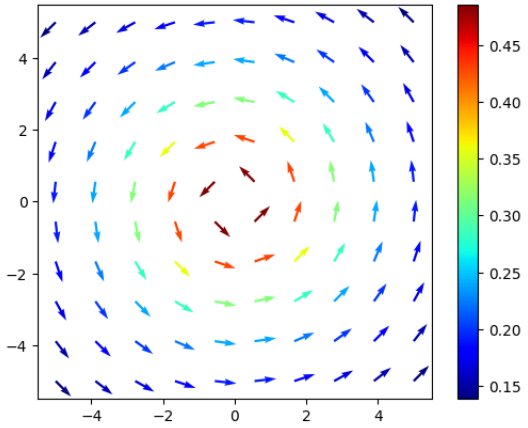
Znázornění v rovině zajíc–rys je také snadné, zadáme veličiny pro vodorovnou a svislou osu a malinko graf vylepšíme přidáním teček, letopočtů a popisku.



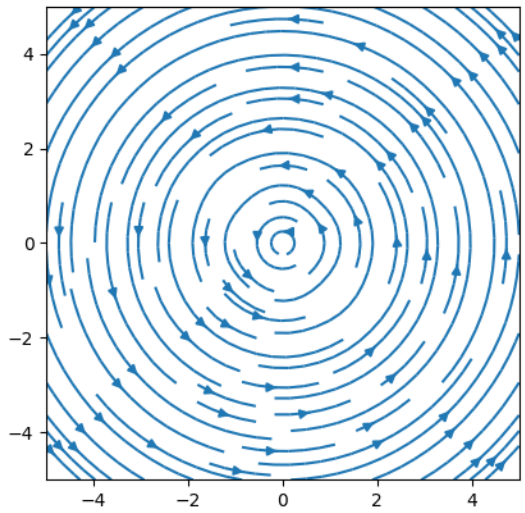
V tomto případě v rovině zajíc–rys vektorová funkce představuje parametricky zadanou křivku, resp. pokud je funkce dána jenom hodnotami z tabulky, potom se křivka redukuje na lomenou čáru. Podobně je možno chápat libovolnou vektorovou funkci jedné proměnné jako parametrickou křivku v prostoru příslušné dimenze.

1.4.3 Vektorové pole

Vektorové pole je funkce, která má na vstupu vektor a na výstupu opět vektor, zpravidla stejné dimenze jako vektor na vstupu. Potom je obvyklé chápat vektor na vstupu jako bod a vektor na výstupu jako vektor s počátkem v uvedeném bodě. Při takovém chápání je poměrně účinné vizualizovat vektorové pole v rovině, tj. funkci, mající na vstupu i na výstupu dvě proměnné. Proměnné na vstupu chápeme jako body v rovině a proměnné na výstupu chápeme jako komponenty orientované úsečky s počátkem v uvažovaném bodě. (Odsud je i název vektorové pole, protože při vizualizaci obdržíme systém orientovaných úseček v rovině. Protože by zpravidla byl výsledný obrázek nepřehledný, délku vektorů můžeme v obrázku sjednotit a informaci o původní délce vyjádřit barevně.



Někdy může být ilustrativnější chápat vektorové pole jako rychlostní pole a vykreslit si příslušný tok.



Tento způsob vizualizace budeme používat při modelování vývoje dvoudruhových populací, například při modelování konkurence nebo vztahu dravce a kořisti. Vektorové pole bude udávat trend, jakým směrem se bude systém vyvíjet a řešením budou parametrické křivky, podobné výše uvedenému příkladu s rysem a zajícem.

KAPITOLA 2

Derivace a integrál

Co se dozvíte v tomto textu

Zásadním problémem v modelování je dovednost vyjádřit rychlost procesu a s touto rychlostí pracovat. Matematice trvalo dlouho, než se tyto dovednosti vypilovaly do formy přijatelné pro použití v přírodních vědách a tato snaha vyústila v partie matematiky nazývané diferenciální a integrální počet. Toto jsou techniky, umožňující pracovat s rychlostí změny funkce a hledat buď rychlost z časového průběhu (diferenciální počet, derivace) nebo časový průběh z rychlosti (integrální počet, integrál). S oběma technikami se seznámíme v následujícím textu.

Na rozdíl od běžně užívaných textů věnovaných této problematice se nebudeme zaměřovat na počítání příkladů na derivace a integrály, ale zdůrazníme si koncept, hlavní myšlenky a využití při formulaci matematických modelů.

Jako ukázku si představíme model ochlazování. Ten je možné použít pro modelování žádoucího ochlazování horké kávy i nežádoucího ochlazování těl živočichů v polárních oblastech. To jsou nejtypičtější začátečnické úlohy z modelování. Jedná se však o univerzální přístup, pomocí kterého později budeme schopni popisat i procesy probíhající v přírodě a ekosystémech.

Foto: Derivace jsou základním nástrojem pro modelování systémů s deterministickým chováním. Jeden z nejjednodušeji modelovatelných procesů je model tepelné výměny. Pochopením tohoto procesu je možné odhalit, jak se živočichové žijící na sněhu a ledu brání teplotním ztrátám při kontaktu končetin se zemí. Zdroj: https://pxhere.com/en/photo/954181

Ústředním tématem přednášky je pojem derivace. Neformálně řečeno je to nástroj, který nám umožňuje pracovat s rychlostí změny funkce. Objevení derivací mělo obrovský dopad v přírodních vědách. Derivace jsou základním nástrojem pro práci s měnícími se veličinami. Jsou tedy přirozeným nástrojem pro práci s dynamickými systémy charakterizovanými proměnnými veličinami a vystavenými proměnným podmínkám.

Každý zvládne jednoduše vypočítat následující úkoly.

- Teplota roste konstantní rychlostí a za tři hodny naroste z teploty $10^\circ\mathrm{C}$ na $16^\circ\mathrm{C}$. Jak rychle teplota roste? (Evidentně rychlostí $2^\circ\mathrm{C}/\mathrm{hod}$).
- Teplota roste konstantní rychlostí $2^\circ\mathrm{C}/\mathrm{hod}$. O kolik teplota naroste za čtyři hodiny? (Evidentně o $8^\circ\mathrm{C}$).

Tyto příklady jsou extrémně jednoduché díky předpokladu, že teplota roste konstantní rychlostí. Potom určujeme rychlost podílem změny teploty a času a změnu určujeme součinem rychlosti a času. V této přednášce si ukážeme nástroje, které umí dávat podobné výsledky i pro procesy probíhající proměnlivou rychlostí.

2.1 Průměrná rychlost a okamžitá rychlost

Budeme se zajímat o to, jak rychle se mění funkční hodnoty v čase nebo obecněji, jak se mění při změnách vstupních dat.

Průměrnou rychlost určujeme tak, že změnu sledované veličiny přepočteme na jednotku času (u závislosti na čase), délky (u závislosti na poloze) nebo obecně na jednotku veličiny, na které sledovaná veličina závisí.

Průměrná rychlost s jakou se mění funkce f na intervalu $[x, x+h]$ je dána vztahem

$$\frac{f(x+h)-f(x)}{h}.$$

Pozitivum tohoto vzorce je, že používá jenom dvě ze čtyř základních operací, rozdíl a dělení. Vzorec s podílem však má i nedostatky. Výrazným nedostatkem je, že průměrná rychlost pracuje jenom s informací v koncových bodech intervalu a proto bohužel neobsahuje informaci, co přesně se děje uvnitř intervalu, přes který průměrujeme. Počítáme-li ale průměr přes stále kratší interval, nevýhoda průměrné rychlosti mizí. Naším cílem bude počítat průměr přes interval prakticky nerozlišitelný od nuly. To by dalo okamžitou rychlost.

Pokud průměrujeme za stále kratší čas, čitatel i jmenovatel se blíží k nule a jsou potíže s interpretací zlomku. Nulou totiž není možné dělit. Musíme vytvořit koncept, který umožní sledovat, co se děje s funkčními hodnotami funkce, pokud se vstupními daty jdeme ke kraji definičního oboru. K tomu použijeme pojem limita. Budeme se (zatím) soustředit na tzv. vlastní limitu ve vlastním bodě. Tím se oproti obecnému postupu mnohé usnadní. Zejména pojem limity můžeme opřít o pojem spojitost, který je přece jenom intuitivnější než obecný koncept využívající limitu jako základní pojem.

2.2 Spojitost

Definice spojitosti zavádí jakousi třídu funkcí, které jsou v jistém smyslu pěkné a můžeme pro ně použít postupy, které pro obecné funkce nefungují. Jsou zde funkce, jejichž funkční hodnoty se mění plynule a nemůžou se změnit skokově. Malá změna ve vstupních datech vyvolá malou změnu ve funkčních hodnotách.

Definice (Okolí)

Okolím bodu x_0 rozumíme libovolný otevřený interval obsahující bod x_0.

Definice (Spojitost)

Buď $f\colon \mathbb{R} \to \mathbb{R}$ funkce jedné proměnné. Řekneme, že funkce f je *spojitá v bodě* x_0 jestliže je v tomto bodě definovaná a pro libovolnou předem zadanou toleranci (i extrémně malou) existuje okolí bodu x_0 takové, že všechny body z okolí bodu x_0 mají funkční hodnotu v rámci uvažované tolerance nerozlišitelnou od $f(x_0)$. Řekneme, že funkce f je *spojitá* na otevřeném intervalu, je-li spojitá v každém jeho bodě.

Definice spojitosti sice není zcela názorná, ale následující definice a věta velmi pomůže. Zhruba řečeno vysvětlují, proč si v naprosté většině prakticky využitelných případů můžeme spojitost ověřit jenom tím, že zjistíme, zda je funkce definována.

Definice (Elementární funkce)

Všechny mnohočleny, goniometrické, cyklometrické, exponenciální a logaritmické funkce a obecná mocnina se nazývají *základní elementární funkce* Všechny funkce, které ze základních elementárních funkcí získáme konečným počtem operací sčítání, odečítání, násobení, dělení a skládání těchto funkcí navzájem se nazývají *elementární funkce*.

Věta (Spojitost elementárních funkcí)

Všechny elementární funkce jsou spojité v každém vnitřním bodě svého definičního oboru.

Podobně jako spojitost funkce jedné proměnné je definována spojitost funkcí více proměnných. Zůstane dokonce v platnosti předchozí věta. V naprosté většině základních praktických aplikací vystačíme s popisem pomocí elementárních funkcí a proto jsou funkce, se kterými pracujeme, zpravidla automaticky spojité. Opatrnost je nutná pouze tam, kde bychom se od elementárních funkcí odchýlili, například při použití nekonečných řad.

Poznámka (Body nespojitosti)

Body, v jejichž okolí je funkce ohraničená, ale je zde porušena spojitost, se nazývají body nespojitosti. Typickými příklady jsou skok a odstranitelná nespojitost. (Opět se vyhneme přesné definici, ale název a vzhled grafu situaci dostatečně ozřejmí.)

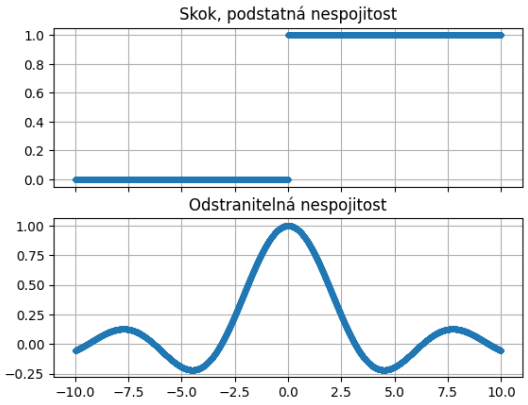
skok : Náhlá změna v jediném bodě, přičemž vlevo i vpravo od tohoto bodu je funkce spojitá. Na jeho odhalení stačí zvolit toleranci v definici spojitosti menší, než je výška skoku. Například $f(x) = \frac{|x|+x}{2x}$ je jednotkový skok v nule.

odstranitelná nespojitost : Chybějící funkční hodnota. Tato nespojitost nás zajímá nejvíce. Je to nespojitost, kterou je možno odstranit vhodným dodefinováním funkční hodnoty v bodě nespojitosti. Například funkce

$$f(x) = \begin{cases} \frac{\sin x}{x} & x \neq 0 \\ 1 & x = 0 \end{cases}$$

je spojitá funkce. Vznikla doplněním jedné funkční hodnoty do definice funkce $\frac{\sin x}{x}$, která má odstranitelnou nespojitost v bodě $x = 0$.

Grafy funkcí z předchozí poznámky jsou níže. Jsou vykresleny z jednotlivých bodů a tyto body nejsou spojeny obvyklou čarou. Tím vyniknou případné nespojitosti.



2.3 Limita

Definici limity opřeme o pojem spojitosti. V podstatě pod limitu skryjeme buď funkční hodnotu spojité funkce (pokud existuje), nebo hodnotu, která danou funkci učiní spojitou. Můžeme tedy limitu považovat za „nejlepší rozumnou náhradu“ funkční hodnoty v tom smyslu, že po předefinování jedné funkční hodnoty se funkce stane spojitou, tj. relativně pěknou.

Definice (Limita)

Nechť f je funkce definovaná v okolí bodu x_0, s případnou výjimkou bodu x_0. Řekneme, že funkce f má v bodě x_0 *limitu* rovnu číslu L, jestliže funkce $g(x)$ definovaná vztahem

$$g(x) = \begin{cases} L & x = x_0 \\ f(x) & \text{jinak,} \end{cases}$$

je spojitá v bodě x_0. Píšeme

$$\lim_{x \to x_0} f(x) = L.$$

Velmi stručně řečeno: pokud se nedá nějaké číslo do funkce dosadit přímo, mohlo by to jít pomocí limity. Například funkce

$$\frac{\sin x}{x}$$

není definována v nule. V okolí nuly se však chová v jistém smyslu pěkně: má funkční hodnoty prakticky nerozlišitelné od jedničky, viz graf v odstavci věnovanému spojitosti. Proto platí

$$\lim_{x \to 0} \frac{\sin x}{x} = 1.$$

2.4 Definice derivace

Nyní jsme připraveni ve vzorci

$$\frac{f(x+h) - f(x)}{h}$$

pro průměrnou rychlost změny na intervalu délky h položit $h = 0$ a dosáhnout toho, že vzorec bude udávat okamžitou rychlost. Protože bychom však přímým dosazením dostali nedefinovaný výraz s nulou ve jmenovateli, uděláme toto dosazení v limitním smyslu.

Definice (Derivace)

Derivací funkce f v bodě x rozumíme limitu

$$\frac{\mathrm{d}f}{\mathrm{d}x} := \lim_{h \to 0} \frac{f(x+h) - f(x)}{h},$$

pokud tato limita existuje.

Derivaci funkce f v bodě x_0 označujeme $f'(x_0)$ nebo $\frac{\mathrm{d}f(x_0)}{\mathrm{d}x}$ nebo $\frac{\mathrm{d}f}{\mathrm{d}x}(x_0)$. Derivaci v libovolném bodě potom f', $f'(x)$ nebo $\frac{\mathrm{d}f}{\mathrm{d}x}$. Zápis $\frac{\mathrm{d}f}{\mathrm{d}x}$ je Leibnizova notace, zápis f' je Lagrangeova notace. Derivace podle času se označuje tečkou, například $\dot{x}$ (Newtonova notace). V některých oborech se používá Eulerova notace Df, $D_x f$ nebo $(Df)(x)$.

Poznámka (Slovní interpretace definice derivace)

Ještě jednou rozšifrujeme jednotlivé komponenty definice derivace.

- Výraz z čitatele, tj. $f(x+h) - f(x)$, je změna veličiny f na intervalu $[x, x+h]$. Často označujeme též Δf.
- Podíl, tj. $\frac{f(x+h)-f(x)}{h}$ je změna veličiny f na intervalu $[x, x+h]$ přepočítaná na jednotku veličiny x, tj. v jistém smyslu průměrná rychlost na tomto intervalu. Často označujeme též $\frac{\Delta f}{\Delta x}$.
- Limita v definici derivace stahuje délku intervalu, na kterém počítáme průměrnou rychlost, k nule. Tím se z průměrné rychlosti stane okamžitá rychlost.

Část definičního vztahu	Slovní interpretace
$f(x)$	funkční hodnota v bodě
$f(x+h)$	funkční hodnota ve vedlejším bodě
$f(x+h)-f(x)$	změna funkce na intervalu $[x, x+h]$
$\frac{f(x+h)}{h}$	průměrná rychlost změny funkce na intervalu $[x, x+h]$, též změna funkce po přepočtu na interval jednotkové délky
$\lim_{h\to 0} \cdots$	limita pro redukci průměrné rychlosti na okamžitou
$\lim_{h\to 0} \frac{f(x+}{}$	okamžitá rychlost změny funkce v bodě x, derivace

Pro interpretaci derivace v praktických úlohách je vodítkem i fyzikální jednotka derivace. Ta je dána následující poznámkou.

Poznámka (Jednotka derivace)

Jednotka derivace $\frac{\mathrm{d}f}{\mathrm{d}x}$ funkce $f(x)$ je stejná, jako jednotka podílu $\frac{f(x)}{x}$.

2.5 Využití derivace

Interpretace derivace v nematematických disciplínách je okamžitá rychlost s jakou veličina f reaguje na změny veličiny x. Často studujeme veličiny závislé na čase s v tomto případě jde tedy o rychlost, s jakou se veličina mění v čase.

Obecně, ať již je nezávislou proměnnou čas či jiná veličina, se derivace $f'(x)$ často slovně interpretuje jako změna veličiny f, odpovídající změně veličiny x o jednotku. Je to podobné, jako údaj o rychlosti na tachometru v automobilu. Ten udává, kolik kilometrů ujedeme za hodinu. Od skutečně uražené dráhy se tento údaj může lišit, protože pohyb může trvat třeba jenom deset minut. A kdyby jízda opravdu trvala hodinu, mohlo by vlivem jízdy v zácpě dojít k podstatnému nesouladu se skutečně uraženou dráhou. Přesto je okamžitá rychlost ukazovaná na tachometru při jízdě automobilem užitečná veličina a nemáme problémy s jejím chápáním. Stejně tak pohlížejme na derivaci.

Derivace (její existence a znaménko) úzce souvisí s některými vlastnostmi funkce. Přesněji, existence derivace implikuje spojitost a znaménko derivace určuje druh monotonie funkce. Přesněji viz následující dvě věty.

Věta (Existence derivace implikuje spojitost)

Má-li funkce f derivaci na intervalu I, je na tomto intervalu spojitá.

Věta (Znaménko derivace implikuje monotonii)

- Má-li funkce f kladnou derivaci na intervalu I, je na tomto intervalu rostoucí.
- Má-li funkce f zápornou derivaci na intervalu I, je na tomto intervalu klesající.

Derivace funkce	Chování funkce
Derivace je nulová.	Funkce je konstantní. Sledovaná veličina se nemění při změně vstupních dat.
Derivace je kladná.	Funkce roste. Pokud data na vstupu rostou, sledovaná veličina také roste.
Derivace je záporná.	Funkce klesá. Pokud data na vstupu rostou, sledovaná veličina klesá.
Derivace je numericky malá (blízká k nule).	Funkce se mění pomalu. Sledovaná veličina reaguje na změny ve vstupních datech pouze málo.
Derivace je numericky velká (hodně kladná nebo hodně záporná).	Funkce se mění rychle. Malá změna na vstupu má velký vliv na sledovanou veličinu.
Derivace je konstantní.	Funkce je lineární. Klesá nebo roste pořád stejně rychle. Pokud vstup roste aritmetickou řadou (po stejných skocích), sledovaná veličina roste nebo klesá také aritmetickou řadou.
Derivace roste.	Funkce je nelineární a roste stále rychleji. Pokud je funkce kladná, rostoucí derivace znamená, že růst se stále zrychluje.
Derivace klesá k nule.	Funkce je nelineární a přibližuje se k vodorovné asymptotě. Pokud je funkce kladná, k nule klesající derivace znamená, že růst se stále zpomaluje a zastaví se.

2.6 Derivace v populační ekologii

V populační ekologii derivaci používáme k vyjádření rychlosti růstu nebo poklesu velikosti sledované populace. Je-li velikost populace jako funkce času dána funkcí $N(t)$, je derivace této funkce, tj.

$$\frac{\mathrm{d}N}{\mathrm{d}t}$$

rychlost růstu této populace. (Záporná derivace tj. záporná rychlost růstu je interpretována jako pokles.) Jednotka derivace je odvozena od povahy úlohy a jednotek sledovaných veličin. Například stovky jedinců za rok.

Ekvivalentním vyjádřením rychlosti je nárůst velikosti populace za jednotku času.

Často pracujeme i s relativní rychlostí růstu, kdy rychlost růstu vztáhneme na jednotkové množství, tj. pracujeme s funkcí

$$\frac{1}{N}\frac{\mathrm{d}N}{\mathrm{d}t},$$

která se někdy v literatuře zapisuje ve formě

$$\frac{\mathrm{d}N}{N\mathrm{d}t}$$

a zpravidla se o ní mluví jako o *specifické rychlosti* růstu, nebo jako o rychlosti růstu *per capita*.

2.7 Model tepelné výměny

Derivace jsou přirozeným jazykem pro formulaci matematického modelu mnoha dějů v přírodě, kdy potřebujeme pracovat s rychlostí a tato rychlost není během děje konstantní. To znamená že změnu není možné určit jednoduše jako součin rychlosti a času. Typickým příkladem je tepelná výměna. Rychlost vyrovnávání teplot souvisí s teplotním rozdílem a během vyrovnávání teplot se tento teplotní rozdíl snižuje. Tím se zpomaluje i dynamika děje.

Horké těleso o teplotě T je v chladnější místnosti o teplotě T_∞. Z fyziky je známo (Newtonův zákon tepelné výměny), že rychlost s jakou klesá teplota tělesa je úměrná teplotnímu rozdílu. Tento rozdíl je $T - T_\infty$ (od většího odečítáme menší).

- Veličina T je teplota tělesa měřená například ve stupních Celsia.
- Veličina t je čas měřený například v hodinách.
- Derivace $\frac{\mathrm{d}T}{\mathrm{d}t}$ ve stupních Celsia za hodinu je rychlost, s jakou roste teplota tělesa.
  - Pokud je například derivace kladná a rovna hodnotě 5 stupňů Celsia za hodinu, znamená to, že teplota roste rychlostí 5 stupňů Celsia za hodinu.
  - Pokud je například derivace záporná a rovna hodnotě -5 stupňů Celsia za hodinu, znamená to, že teplota klesá rychlostí 5 stupňů Celsia za hodinu.
  - Pokud je derivace dána vztahem $-e^{-t}$, kde t je čas v hodinách a derivace vychází ve stupních Celsia za hodinu, využijeme toho, že $e^0 = 1$ a $e^{-1} = 0.37$. To znamená, že na počátku se teplota snižuje okamžitou rychlostí jeden stupeň Celsia za hodinu, tato rychlost ochlazování se pozvolna mění a například po hodině se teplota snižuje už jenom rychlostí 0.37 stupně Celsia za hodinu.
- Matematickým vyjádřením toho, že rychlost s jakou se mění teplota je úměrná teplotnímu rozdílu $T - T_\infty$ je rovnice

  $$\frac{\mathrm{d}T}{\mathrm{d}t} = -k(T - T_\infty),$$

  kde k je konstanta úměrnosti a záporné znaménko vyjadřuje, že teplota klesá. Konstanta k je číselně rovna rychlosti ochlazování v situaci, kdy je jednotkový rozdíl mezi teplotou objektu a okolí.
- Neznámou v sestavené rovnici je funkce a v rovnici figuruje derivace této funkce. Takové rovnice se nazývají diferenciální rovnice.
- Uvedená rovnice udává scénář vývoje systému. Je to jakési kvantitativní vyjádření funkce mechanismu, který vyrovnává teploty. V praxi je nutné ještě k plnohodnotné simulaci zadat výchozí stav. Většinou je výchozím okamžikem čas nula a proto se zadává výchozí stav podmínkou

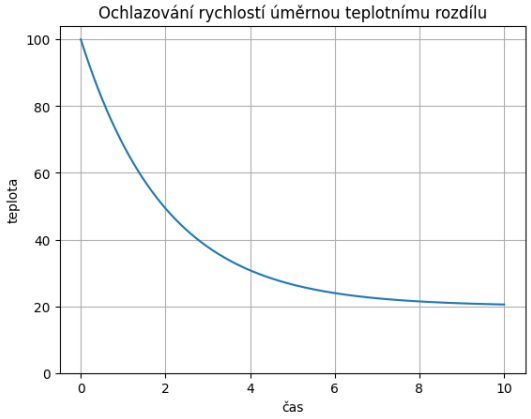
  $$T(0) = T_0,$$

  kde T_0 je počáteční teplota. Tato podmínka se nazývá počáteční podmínka.

Rada: Předchozí příklad je často v různých obměnách používán na modelování ochlazování kávy, což je proces, který většina lidí důvěrně zná. Nemáme pochopitelně ambice se domnívat, že bychom dokázali z této rovnice odvodit nějaké zásadní výsledky aplikovatelné při pití ranní kávy. Učíme se na malých věcech, abychom později mohli dělat věci velké. Na známých věcech se učíme aparát, který bude naším jediným nástrojem tam, kde intuice začne selhávat. Z tohoto příkladu je nutné si odnést, že derivace, jako rychlost změny, hraje roli při kvantitativním popisu dějů a při studia procesů, kdy se mění veličiny. Ať už doopravdy (studium pohybu nebo dějů, probíhajících v čase) nebo virtuálně (problémy spojené

s mechanikou, včetně statiky, stability a deformací, často pracují s virtuálními změnami, tj. se změnami, které jsou sice z hlediska úlohy přípustné, ale příroda je z nějakého důvodu nerealizuje). Tedy naprostá většina dějů a jevů, které studujeme a chceme jim rozumět. Jakmile se v popisu fyzikálního zákona nebo přírodního procesu objeví slovo *rychlost*, někdy nahrazené souslovím *časová změna*, znamená to, že kvantitativní popis se děje pomocí derivací.

V této chvíli je pro nás cenné to, že umíme přeformulovat fyzikální popis vývoje (rychlost změny teploty je úměrná rozdílu teplot) na kvantitativní popis, kde dokážeme realizovat numerickou simulaci. Realizace takové simulace může vypadat například tak, že na krátký časový krok budeme předpokládat konstantní rychlost. Tuto rychlost použijeme pro odhad nové teploty, tato nová teplota změní teplotní rozdíl, tím se změní i rychlost a postup opakujeme.



2.8 Analytický výpočet derivace

Analytickým výpočtem derivace rozumíme získání vzorce pro derivaci funkce z analytického předpisu funkce, tj. ze vzorce definujícího funkční hodnoty. Toto se naučíme později v obecnějším kontextu, až se budeme věnovat derivacím funkcí více proměnných, parciálním derivacím.

Důležité: Nikdy (nebo alespoň skoro nikdy) nederivujeme pomocí definice, ale používáme vzorce pro derivace základních elementárních funkcí a pro derivace matematických operací s funkcemi. Další možností je využití systémů počítačové algebry, které umožňují pracovat se symbolickými výrazy namísto numerických dat. V programu Python je možno použít knihovnu SymPy.

Při analytickém výpočtu derivace používáme pravidla pro derivování základních elementárních funkcí a pravidla pro derivaci matematických operací. Nám budou stačit pouze základní vzorce, které uvedeme nejprve slovně.

- Derivace konstantní funkce je nulová funkce.
- Derivace součtu funkcí je součet derivací těchto funkcí.
- Derivace konstantního násobku funkce je konstantní násobek derivace této funkce.
- Derivace složené funkce je součinem derivace vnější a vnitřní složky.
- Derivace funkce x je rovna jedné.
- Derivace funkce e^x je e^x.

Tedy pro funkce f a g proměnné x a konstantu c platí následující vzorce.

$$\frac{\mathrm{d}c}{\mathrm{d}x} = 0$$
$$\frac{\mathrm{d}}{\mathrm{d}x}(f+g) = \frac{\mathrm{d}f}{\mathrm{d}x} + \frac{\mathrm{d}g}{\mathrm{d}x}$$
$$\frac{\mathrm{d}}{\mathrm{d}x}(cf) = c\frac{\mathrm{d}f}{\mathrm{d}x}$$
$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}g}\frac{\mathrm{d}g}{\mathrm{d}x}$$
$$\frac{\mathrm{d}}{\mathrm{d}x}(x) = 1$$
$$\frac{\mathrm{d}}{\mathrm{d}x}(e^x) = e^x$$

2.9 Numerický výpočet derivace

Numerickým výpočtem derivace rozumíme výpočet číselné hodnoty derivace funkce z funkčních hodnot vypočtených ve vhodných bodech definičního oboru funkce, zpravidla v rovnoměrně rozmístěných bodech na intervalu konečné délky. Tento výpočet může být a bývá zatížen zaokrouhlovací chybou.

Numerický výpočet se realizuje v případě, že není k dispozici analytický vzorec pro funkci, nebo není efektivní či nutné využít analytický postup. Provádí se zpravidla tak, že vstupem je funkce definovaná v bodech rovnoměrně rozložených na jejím definičním oboru a namísto derivace jako okamžité rychlosti počítáme průměrnou rychlost. K tomu se využijí sousední body a odhad pro derivaci v bodě x poté je

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^2),$$

kdy asymptotika je uvažována v okolí nuly. Tento vzorec se nazývá *centrální diference* s krokem h. V krajních bodech definičního oboru nemáme k dispozici oba sousední body a proto se používá k výpočtu dopředná a zpětná diference dané po řadě vztahy

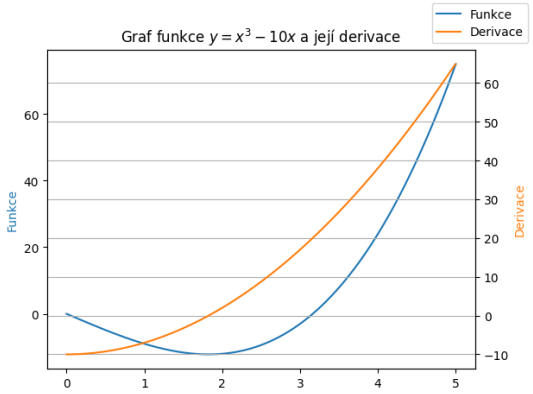
$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h)$$

a

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{f(x) - f(x-h)}{h} + \mathcal{O}(h).$$

Všimněte si, že centrální diference je o řád přesnější než dopředná nebo zpětná diference. Pro $h = 0.01$ jsou $\mathcal{O}(h)$ řádově setiny, kdežto $\mathcal{O}(h^2)$ jsou řádově desetitisíciny.

Graf funkce a její derivace je na následujícím obrázku. Mřížka je nakreslena pro pravou osu, protože funkční hodnoty funkce (škála na levé svislé ose) nejsou podstatné a derivaci nevlivní. Derivace nesleduje číselné hodnoty jako takové, ale sleduje, jak rychle se tyto hodnoty mění. Funkce nejprve klesá, její derivace je záporná. V minimu se pokles obrátí v růst a derivace se změní ze záporné na kladnou. V minimu je tedy derivace rovna nule. Poté funkce roste a roste sále rychleji. To znamená, že derivace také roste.



2.10 Integrál a jeho využití

Naučili jsme se pracovat s derivacemi, tedy s rychlostí změny. Známe-li funkci a zderivujeme ji, dostaneme rychlost změny. Pokud potom původní funkci „ztratíme“ a zůstane nám jenom derivace, je otázka, jestli dokážeme původní funkci z této derivace najít. Odpověď zní, že v jistém smyslu ano. Spojení „v jistém smyslu“ naznačuje, že souvislost nebude tak snadná jako je souvislost u navzájem inverzních funkcí. Derivováním totiž můžeme ztratit aditivní konstanty, které v derivaci dávají nulu a zpětně není možné rekonstruovat, derivováním jaké konstanty jsme tuto nulu dostali.

Definice (Neurčitý integrál)

Řekneme, že funkce F je *primitivní funkcí* k funkci f na intervalu I, jestliže platí

$$F'(x) = f(x)$$

na intervalu I. Množina všech primitivních funkcí k funkci f se nazývá *neurčitý integrál* funkce f a značí

$$\int f(x)\,\mathrm{d}x.$$

Platí následující.

- Ke každé spojité funkci existuje neurčitý integrál.
- Primitivní funkce je dána jednoznačně až na aditivní konstantu.

Poznámka 2.10.1 (Veličina vypočtená z rychlosti své změny)

Pokud se veličina $f(t)$ mění v čase rychlostí $r(t)$, platí

$$f(t) = \int r(t)\,\mathrm{d}t,$$

přičemž pravá strana je dána jednoznačně až na aditivní konstantu. To koresponduje s pozorováním, že rychlost změn k jednoznačné identifikaci časového průběhu měnící se veličiny nestačí. Je potřeba mít zadán ještě výchozí stav.

Neurčitý integrál není dán jednoznačně. Proto je někdy výhodnější počítat pouze změnu měnící se veličiny, tj. určovat rozdíl na konci a na začátku intervalu. To vede na poněkud odlišné pojetí integrálu.

Definice (Newtonův určitý integrál)

Buď f funkce a F její primitivní funkce na intervalu I. Buď $[a, b] \subset I$ podinterval v I. *Určitým integrálem funkce f na intervalu $[a, b]$* rozumíme veličinu označenou a definovanou vztahem

$$\int_a^b f(x)\mathrm{d}x := F(b) - F(a).$$

Poznámka 2.10.2 (Změna veličiny vypočtená pomocí rychlosti)

Pokud se veličina $f(t)$ mění v časovém intervalu od $t = a$ do $t = b$ rychlostí $r(t)$, je změna veličiny f za tento časový okamžik rovna

$$\Delta f = f(b) - f(a) = \int_a^b r(t)\,\mathrm{d}t.$$

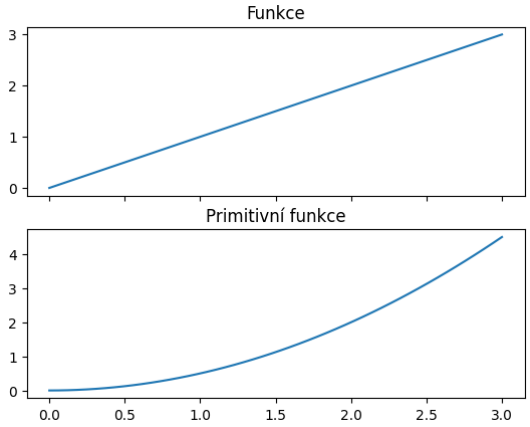
- Určitý integrál je možno numericky aproximovat i bez znalosti primitivní funkce. Je možno použít funkci `numpy.trapz`, `scipy.integrate.trapezoid` pokud je funkce dána pomocí hodnot, nebo `scipy.integrate.quad`, pokud je funkce dána funkčním předpisem.
- Neurčitý integrál je možno vypočítat pomocí určitého integrálu jako funkci horní meze.

$$\int f(x)\,\mathrm{d}x = \int_0^x f(t)\,\mathrm{d}t + C$$

Následující sada příkazů vypočítá pro zadanou funkci $f(x)$ primitivní funkci jako integrál

$$\int_0^x f(t)\,\mathrm{d}t$$

a funkci i její integrál vykreslí. Výpočet zajišťuje funkce `scipy.integrate.cumulative_trapezoid`. Kromě toho je možné počítat integrály i analyticky použitím knihovny `sympy`, to však v našich výpočtech potřebovat nebudeme.



Integrál je tedy možno použít k nalezení funkce, pokud víme, jak rychle se tato funkce mění v čase. Bohužel naprostá většina procesů probíhajících v přírodě souvisí ne s časem, ale s velikostí měnící se veličiny. Například rychlost s jakou roste populace jelenů souvisí s velikostí této populace (větší populace jelenů má více plodných laní) a ne s časem. (V roce 2020 roste stejná populace stejně rychle jako v roce 2016.) Viděli jsme to i na modelu tepelné výměny, kdy rychlost změny teploty nesouvisela přímo s časem, ale souvisela s dosaženou teplotou. V takových případech je nutné použít speciální aparát nazývaný diferenciální rovnice, který si představíme v další přednášce.

KAPITOLA 3

Diferenciální rovnice

Co se dozvíte v tomto textu

V tomto textu si ukážeme, jak umí věda pracovat s modely spojujícími rychlost měnící se veličiny a hodnotu této veličiny. Zpravidla se jedná o rovnice obsahující neznámou ve formě funkce a v rovnici je i derivace této funkce. Tyto rovnice se nazývají diferenciální rovnice a jsou nejběžnějším modelovacím nástrojem napříč mnoha obory, od biologie přes ekologii a chemii až k fyzice.

Jako ukázku využití si představíme jeden z nejstarších ale dodnes užívaných ekologických modelů, model ostrovní biogeografie slavných ekologů McArthura a Wilsona. Tento model se snaží vysvětlit odlišnosti v ekosystémech na ostrovech a najít podstatu těchto odlišností.

Foto: Jednou z ukázek specifik ostrovních ekosystémů je vzácná endemická liška ostrovní, Urocyon littoralis. Žije na ostrovech u pobřeží Kalifornie. Autor Shanthanu Bhardwaj, https://wikimedia.org

Model ochlazování kávy z minulé přednášky byl reprezentantem nové třídy úloh, nazývaných diferenciální rovnice. Roli v něm hrála mimo jiné rychlost změny teploty, tedy derivace teploty podle času. Protože se jedná o derivaci funkce jedné proměnné, jak jsme se s ní seznámili dříve, říká se těmto rovnicím obyčejné diferenciální rovnice. V průběhu semestru se setkáme ještě s dalším typem derivací, parciálními derivacemi. Při jejich použití v matematických modelech dostáváme parciální diferenciální rovnice.

3.1 Obyčejná diferenciální rovnice

Obyčejná diferenciální rovnice je rovnice, kde vystupuje neznámá funkce a její derivace. Setkáváme se s ní například všude tam, kde rychlost růstu nebo poklesu veličiny souvisí s její velikostí. Například rychlost s jakou se mění teplota horké kávy je funkcí teploty samotné. Rychlost tepelné výměny mezi dvěma tělesy je totiž úměrná rozdílu jejich teplot (Newtonův zákon). Takto se přirozeně diferenciální rovnice objevují v modelech mnoha dějů a jevů. Podstatu děje, který modelujeme, musí dodat fyzika, biologie nebo jiná aplikovaná věda. To v matematice obsaženo není. Matematika poté poslouží k analýze, jaké jsou pozorovatelné důsledky a tím se ověří, jestli příslušná aplikovaná věda správně vystihuje podstatu modelovaného děje.

Definice (Diferenciální rovnice)

Obyčejnou diferenciální rovnicí prvního řádu rozřešenou vzhledem k derivaci (stručněji též diferenciální rovnicí, DR) s neznámou x rozumíme rovnici tvaru

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \varphi(t, x) \tag{3.1}$$

kde φ je funkce dvou proměnných.

Další formy zápisu rovnice (3.1) jsou následující.

$$x' = \varphi(t, x)$$

$$\mathrm{d}x = \varphi(t, x)\mathrm{d}t$$

$$\mathrm{d}x - \varphi(t,x)\mathrm{d}t = 0$$

Diferenciální rovnice bývá v aplikacích matematickým modelem kvantifikujícím scénář vývoje systému. Řešením jsou všechny možnosti, jak se tento systém může vyvíjet. K jednoznačnému předpovězení budoucího stavu je ovšem nutno znát také stav počáteční, který ze všech teoreticky možných průběhů vybere průběh odpovídající modelované situaci. Tento stav vyjadřuje počáteční podmínka, uvedená v následující definici.

Definice (Počáteční podmínka, Cauchyova úloha)

Nechť t_0, x_0 jsou reálná čísla. Úloha najít řešení rovnice (3.1), které splňuje zadanou *počáteční podmínku*

$$x(t_0) = x_0 \tag{3.2}$$

se nazývá *počáteční* (též *Cauchyova*) *úloha*.

Řešení Cauchyovy úlohy nazýváme též *partikulárním řešením rovnice*. Graf libovolného partikulárního řešení se nazývá *integrální křivka*.

Příklad (praktická interpretace řešení počáteční úlohy).

- Pokud diferenciální rovnice udává rychlost ochlazování horkého nápoje a počáteční podmínka teplotu na počátku, je řešením počáteční úlohy funkce, do které dosadíme čas a přímo dostáváme teplotu nápoje v daném čase.
- Pokud diferenciální rovnice udává rychlost růstu populace živočišného druhu v čase a počáteční podmínka velikost populace na počátku sledování, je řešením počáteční úlohy funkce, do které dosadíme čas a přímo dostáváme velikost populace v daném čase.

3.2 Existence a jednoznačnost řešení

Jedna diferenciální rovnice má nekonečně mnoho řešení. Zpravidla je dokážeme zapsat pomocí jediného vzorce, který obsahuje nějakou (alespoň do jisté míry libovolnou) konstantu C. Takový vzorec se nazývá **obecné řešení rovnice**. Pokud není zadána počáteční podmínka a mluvíme o **partikulárním řešení**, máme tím na mysli jednu libovolnou funkci splňující diferenciální rovnici.

Pokud se proto bavíme o jednoznačnosti, máme tím na mysli jednoznačnost řešení *počáteční úlohy.*

Věta (Existence a jednoznačnost řešení Cauchyovy úlohy)

Počáteční úloha

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \varphi(t,x), \quad x(t_0) = x_0$$

má právě jedno řešení definované v nějakém okolí počáteční podmínky, jestliže existuje konstanta L taková, že platí

$$|\varphi(t,x_1) - \varphi(t,x_2)| \leq L|x_1 - x_2|$$

pro všechna t, x_1, x_2 z nějakého okolí počáteční podmínky.

Vlastnost z předchozí věty se nazývá Lipschitzovská spojitost. Počáteční úloha má tedy právě jedno řešení, jestliže je funkce v okolí počáteční podmínky Lipschitzovsky spojitá vzhledem ke druhé proměnné. (Picardova–Lindelöfova věta). Věta dokonce umožňuje i najít interval, na kterém je jednoznačnost garantována, viz [10].

3.3 Směrové pole

Derivace funkce v daném bodě udává rychlost růstu této funkce. Geometricky se jedná o směrnici tečny. Díky tomu lze rovnici

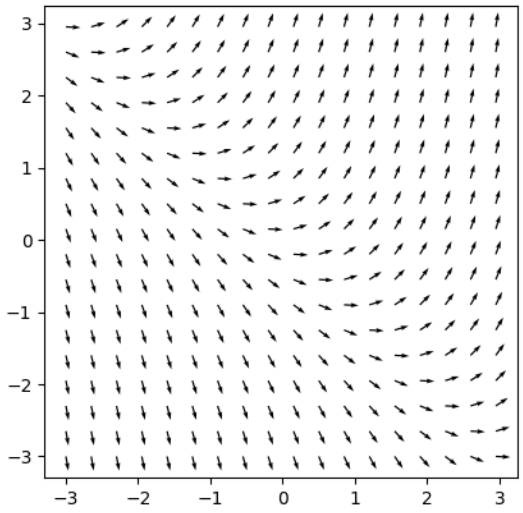
$$\frac{\mathrm{d}x}{\mathrm{d}t} = \varphi(t,x)$$

chápat jako předpis, který každému bodu v rovině (t,x) přiřadí směrnici tečny k integrální křivce, která tímto bodem prochází. Sestrojíme-li v dostatečném počtu (například i náhodně zvolených) bodů $[t,x]$ v rovině vektory $(1,\varphi(t,x))$, obdržíme **směrové pole diferenciální rovnice —** systém lineárních elementů, které jsou tečné k integrálním křivkám.

Počáteční podmínka $x(t_0) = x_0$ geometricky vyjadřuje skutečnost, že graf příslušného řešení prochází v rovině bodem $[t_0, x_0]$. Má-li tato počáteční úloha jediné řešení, neprochází bodem $[t_0, x_0]$ žádná další křivka. Má-li každá počáteční úloha (3.1)-(3.2) jediné řešení (což bude pro nás velice častý případ), znamená to, že integrální křivky se *nikde neprotínají*.

Následující obrázek obsahuje směrové pole rovnice

$$\frac{\mathrm{d}x}{\mathrm{d}t} = x + t.$$



3.4 Analytické a numerické řešení

Při studiu modelu založeného na diferenciální rovnici bývá často výhodné mít k dispozici vzorec, vyjadřující všechny funkce vyhovující rovnici nebo funkci vyhovující rovnici i počáteční podmínce. Takové řešení se nazývá analytické řešení. Například analytickým řešením počáteční úlohy

$$\frac{\mathrm{d}T}{\mathrm{d}t} = -k(T - T_\infty), \quad T(0) = T_0$$

je funkce

$$T(t) = T_\infty + (T_0 - T_\infty)e^{-kt}.$$

V takovém případě máme dokonce k dispozici vzorec obsahující parametry k a T_0 figurující v modelu a toto řešení tedy reprezentuje celou třídu různých úloh.

Nalezení analytického řešení však někdy bývá spojeno s nemalými obtížemi, někdy dokonce s obtížemi principiálně nepřekonatelnými. V takovém případě je možno rovnici vyřešit numericky a najít hodnoty řešení v určitých předem zvolených bodech. Protože diferenciální rovnice jsou základní technikou při modelování přírodních a fyzikálních jevů, jsou k dispozici robustní a časem prověřené techniky a algoritmy, které bývají jako knihovny zařazeny do běžných programů umožňujících práci s těmito rovnicemi.

Výhodou numerického přístupu je, že namísto funkčního předpisu máme řadu numerických hodnot a s čísly se zpravidla pracuje pohodlněji než s funkcemi. Dokonce nejčastější metoda jak získat přehled o chování funkce dané funkčním předpisem spočívá v nakreslení grafu, což v podstatě odpovídá převedení na čísla a znázornění těchto čísel v souřadné soustavě. Na druhou stranu, převedení na čísla může býti nevýhodou. Například nejsme schopni postihnout závislost na parametrech. Numerické simulace se dají dělat pro *konkrétní hodnoty parametrů* a není možné v nich podchytit závislost na parametrech jinak, než řešit rovnici opakovaně pro různé parametry a porovnávat vzniklá řešení.

I v případech, kdy je možné najít analytické řešení, může však být numerické řešení užitečnější. Analytická řešení mohou být tak složitá, že z nich není ani po bližším prozkoumání jasné, jak se systém chová. Proto se nejprve zaměříme na numerické řešení.

3.5 Model ostrovní biogeografie

Životní prostředí podléhá neustálým změnám, ať už vlivem činnosti člověka či vlivem jiných aspektů. V důsledku toho jsou některé živočišné či rostlinné druhy ohroženy, jiné vymírají a jiné se naopak začínají více a více prosazovat. K pochopení tohoto procesu může pomoci i *ostrovní ekologie* - studium vývoje druhů na ostrovech. Z hlediska pevniny jsou totiž ostrovy relativně nestálá a neustále se vyvíjející společenstva, vysoce citlivá na vnější zásahy, na kterých je možno sledovat vývoj jednotlivých druhů – jejich stability, rozmanitosti a pod. Principy ostrovní ekologie se nevztahují pouze na ostrovy v zeměpisném slova smyslu. Jde o jakýkoliv habitat oddělený od okolí. Například vrcholky hor jsou ostrovy v moři stanovišť s menší nadmořskou výškou, parky jsou ostrovy zeleně v moři městské zástavby, lesy jsou ostrovy v zemědělsky využívané krajině, živočichové jsou ostrovy pro parazity na nich žijící a podobně. Ostrovní ekologie je tedy nedílnou součástí ekologie jako celku i u vnitrozemských států.

R. H. Mac Arthur a E. O. Wilson představili v 60. letech 20. stol. následující teorii dynamické rovnováhy počtu druhů na ostrově. Tato teorie získala veliký ohlas a oba vědce proslavila mezi ekology, protože vysvětlovala fenomény spojené s dynamikou populací na ostrovech, jako souvislost druhové rozmanitosti se vzdáleností a rozlohou ostrova. Vzhledem k možnostem aplikací teorie i na „ostrovy“ v přeneseném smyslu tohoto slova se teorie Mac Arthura a Wilsona stala základním stavebním kamenem moderní krajinné ekologie (viz [11]).

Uvažujme ostrov, nacházející se relativně nedaleko pevniny – takový, že na něj mohou z pevniny migrovat nové druhy (větrem, přes moře, v trusu ptáků a pod.), které na ostrově dosud nežijí. Tyto druhy se na ostrově buď uchytí nebo neuchytí. V případě, že se druh úspěšně uchytí a kolonizuje ostrov, může tato kolonizace být na úkor druhů jiných, které následkem tohoto vymřou. Protože pevnina má mnohem větší nosnou kapacitu než ostrov, slouží jako jistá zásobárna nových druhů pro uvažovaný ostrov a ostrov je tedy neustále pod vlivem imigrace. Protože ostrov má menší nosnou kapacitu, než mnohem rozlehlejší a bohatší pevnina, může na něm trvale žít méně druhů než na pevnině.

Předpokládejme, že rychlost kolonizace, tj. počet druhů, které v čase t proniknou na ostrov a úspěšně se zde zabydlí, roste s počtem imigrantů a klesá s počtem druhů, které na ostrově již žijí. První předpoklad je zcela přirozený, druhý vyjadřuje v ekologii obvyklé tvrzení, že komplexnější společenstva organismů jsou stabilnější a lépe odolávají invazi nových druhů. Počet imigrantů klesá s rostoucí vzdáleností ostrova od pevniny, což je opět přirozený předpoklad. Uvedené předpoklady splňuje funkce

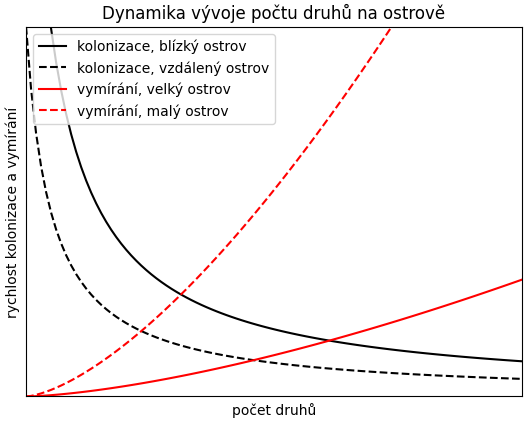
$$\frac{b}{D(N+\beta)},$$

kde N je počet druhů na ostrově v čase t, D je vzdálenost ostrova od pevniny, β je nezáporná a b kladná konstanta. Monografie [3] navrhuje, že rychlost kolonizace souvisí i s rozlohou ostrova. Tato veličina by se dala včlenit například do parametru D, který by potom nevyjadřoval vzdálenost, ale vhodným způsobem by zohledňoval společný vliv vzdálenosti a velikosti ostrova.

Předpokládejme, že rychlost vymírání druhů, které v minulosti již úspěšně kolonizovaly ostrov, ale neobstály v konkurenci pozdějších kolonizátorů, roste s klesající rozlohou ostrova a s rostoucím počtem druhů na ostrově. Tento předpoklad je opět přirozený, vzhledem k tomu, že ostrov menší rozlohy má menší nosnou kapacitu. Kromě toho, byl tento předpoklad prověřen i pokusy (viz [3]). Rychlost vymírání druhů je možné modelovat funkcí

$$a\frac{N^k}{S},$$

kde S je rozloha ostrova a a a k jsou kladné konstanty.



Nyní budeme pohlížet na rychlost s jakou se mění počet druhů na ostrově ze dvou hledisek a tato hlediska nám pomůžou sestavit matematický model pro popis dynamiky ostrova.

- Rychlost s jakou se mění počet druhů na ostrově je derivace počtu druhů podle času. To je přímo význam derivace podle času.

$$\frac{\mathrm{d}N}{\mathrm{d}t}$$

- Rychlost s jakou se mění počet druhů na ostrově je také změna počtu druhů za jednotku času. Tato změna se vypočítá jako rozdíl počtu druhů, které ostrov za jednotku času kolonizovaly a počtu druhů, které na ostrově za tuto dobu vymřely. Tedy půjde o rozdíl rychlosti kolonizace a rychlosti vymírání.

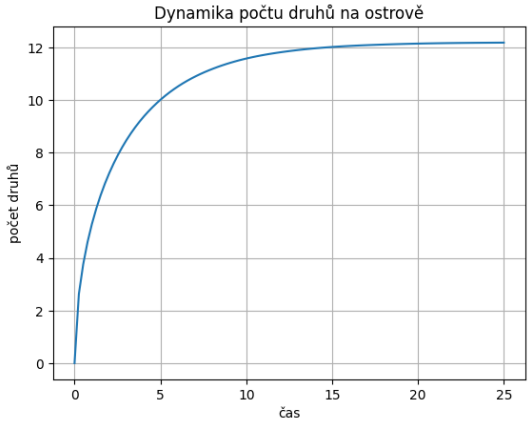
$$\frac{b}{D(N+\beta)} - a\frac{N^k}{S}.$$

V předchozích bodech jsme dvakrát z různých pohledů představili stejnou veličinu a proto se oba výrazy musí rovnat. Počet druhů na ostrově rozlohy S ve vzdálenosti D od pevniny tedy vyhovuje diferenciální rovnici

$$\frac{\mathrm{d}N}{\mathrm{d}t} = \frac{b}{D(N+\beta)} - a\frac{N^k}{S}.$$

Předpokládáme-li, že na počátku byl ostrov neosídlený, připojíme podmínku $N(0) = 0$.

Řešení rovnice pro jistou sadu parametrů je na následujícím obrázku.



Všimněme si, že počet druhů ustálí na konstantní hodnotě. To znamená, že nepřevládá ani vymírání druhů ani kolonizace, ale oba procesy jsou v rovnováze. Tato rovnováha je dynamická. Není nijak garantováno, že druhové složení je neměnné. V praxi dojde k tomu, že druhové bohatství (počet druhů) bude konstantní, bude se však měnit složení druhů.

Východiska Mac Arthurovy a Wilsonovy teorie byla potvrzena pokusem s ostrůvky poblíž Floridy. Ostrůvky byly zbaveny chemickou cestou bezobratlých živočichů. Za necelý rok se druhové bohatství díky invazi z pevniny obnovilo. Konkrétní druhové složení však bylo jiné, než před zásahem, a toto druhové složení se neustále měnilo. Ostrůvky poblíž pobřeží hostily více druhů než ty vzdálenější a při dodatečném umělém snížení velikosti některých ostrůvků se jejich druhové bohatství zmenšilo. [9]

V roce 1883 byl opakovanými sopečnými výbuchy téměř zničen život na ostrově u sopky Krakatoa, který leží cca 25km od Jávy a má rozlohu 20 km^2. Již v roce 1921 byl tento ostrov osídlen 27 druhy ptáků. Tento počet se

v pozdějších letech již neměnil, měnila se pouze druhová skladba. Vzhledem k tomu, že ptáci snadno pronikají na ostrov, se poměrně rychle obnovila jejich rovnováha. Hodnoty koeficientů v tomto případě jsou $\frac{b}{D} = 22\,\text{rok}^{-1}$, $\beta = 1$ a $\frac{a}{S} = 0.03\,\text{rok}^{-1}$. Rostlin bylo na tomto ostrově v roce 1934 celkem 271 druhů, tento počet však nadále rostl. Poněkud paradoxně může jevit fakt, že jako první se na ostrově úspěšně uchytili mrchožraví živočichové. Tento jev je však přirozený, uvědomíme-li si, že tito živočichové měli nejhojnější zdroje potravy v podobě mršin živočichů neúspěšně invadujících druhů. [9]

Základy ostrovní ekologie nacházejí jisté uplatnění i při zakládání a udržování rezervací, za účelem zachování druhové pestrosti. Protože, jak ukazuje ostrovní ekologie, omezené kapacity ostrova a nízká migrace způsobí, že přežívá pouze omezený počet druhů, je nutno „ostrovním efektům“ co nejvíce zabránit. To lze činit například zakládáním rezervací o velké rozloze. Dále je vhodné, aby v rezervaci byl velký počet ekologických nik a remízků, kde se jednotlivé druhy mohou uchytit (zvětší se b a zmenší a). Je-li nutno založit rezervaci o malé rozloze, je nutno ji podrobit velice přísnému režimu ochrany a je vhodné mít systém rezervací, které jsou propojeny migračními cestami.

3.6 Transformace diferenciálních rovnic

Naučíme se vyjadřovat rovnice s derivacemi v jiných proměnných tak, aby bylo možné snížit počet parametrů v této rovnici. Pro jednoduchost budeme uvažovat jenom případ, kdy nová proměnná je lineární funkcí původní proměnné.

Uvažujme funkci y proměnné x. Připomeneme si vzorce pro derivaci součtu, derivaci konstantního násobku a derivaci složené funkce, ale uvedeme si je v kontextu vhodném pro studium diferenciálních rovnic.

- Z derivace součtu a z derivace konstanty plyne pro funkci y a konstantu y_0 vztah

$$\frac{\mathrm{d}(x \pm x_0)}{\mathrm{d}t} = \frac{\mathrm{d}x}{\mathrm{d}t} \pm \frac{\mathrm{d}x_0}{\mathrm{d}t} = \frac{\mathrm{d}x}{\mathrm{d}t} \pm 0 = \frac{\mathrm{d}x}{\mathrm{d}t}.$$

- Z derivace konstantního násobku funkce plyne pro funkci y a konstantu k vztah

$$\frac{\mathrm{d}(kx)}{\mathrm{d}t} = k\frac{\mathrm{d}x}{\mathrm{d}t}.$$

- Z derivace složené funkce plyne pro konstantu k a veličinu $T = kt$ vztah

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \frac{\mathrm{d}x}{\mathrm{d}T}\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{\mathrm{d}x}{\mathrm{d}T}k$$

tj.

$$\frac{\mathrm{d}x}{\mathrm{d}(kt)} = \frac{\mathrm{d}x}{\mathrm{d}T} = \frac{1}{k}\frac{\mathrm{d}x}{\mathrm{d}t}.$$

Výše uvedené výpočty je možno shrnout do pravidla v následující poznámce.

Poznámka (Transformace diferenciální rovnice do jiných jednotek)

Pro $X = k_1(x - x_0)$ a $T = k_2 t$ platí

$$\frac{\mathrm{d}X}{\mathrm{d}T} = \frac{\mathrm{d}\Big(k_1(x - x_0)\Big)}{\mathrm{d}(k_2 t)} = \frac{k_1}{k_2}\frac{\mathrm{d}x}{\mathrm{d}t}.$$

Výraz nalevo neobsahuje konstanty, které jsou ve výrazu napravo. Tyto konstanty jsou v definici nových veličin T a X.

Navíc vzorec z poznámky silně připomíná klasické počítání se zlomky. Proto máme Leibnizův tvar zápisu derivací $\frac{\mathrm{d}X}{\mathrm{d}T}$ při studiu diferenciálních rovnic více v oblibě, než zápis Lagrangeův, x'.

Příklad (Bezrozměrná rovnice tepelné výměny)

Model tepelné výměny

$$\frac{\mathrm{d}T}{\mathrm{d}t} = -k(T - T_\infty), \quad T(0) = T_0$$

obsahuje tři parametry: teplotu okolního prostředí T_∞, počáteční teplotu T_0 a konstantu k související s fyzikálními vlastnostmi prostředí. Postupně můžeme posunout teplotní stupnici tak, aby teplota okolí byla nula a počáteční teplota jedna, tj. hodnotu T snížíme o T_∞ a upravíme dílek stupnice $(T_0 - T_\infty)$-krát

$$\frac{\mathrm{d}\left(\frac{T - T_\infty}{T_0 - T_\infty}\right)}{\mathrm{d}t} = -k\frac{T - T_\infty}{T_0 - T_\infty}$$

vydělit konstantou k

$$\frac{\mathrm{d}\left(\frac{T - T_\infty}{T_0 - T_\infty}\right)}{k\mathrm{d}t} = -\frac{T - T_\infty}{T_0 - T_\infty}$$

a přeškálovat pomocí konstanty k čas

$$\frac{\mathrm{d}\left(\frac{T - T_\infty}{T_0 - T_\infty}\right)}{\mathrm{d}(kt)} = -\frac{T - T_\infty}{T_0 - T_\infty}.$$

Po substituci $y = \frac{T - T_\infty}{T_0 - T_\infty}$, $x = kt$ má úloha tvar

$$\frac{\mathrm{d}y}{\mathrm{d}x} = -y, \quad y(0) = 1.$$

Nová rovnice *neobsahuje žádné parametry* a proto je pro studium jednodušší. Přesto je v ní obsažena veškerá informace obsažená v rovnici původní. Tuto informaci je

však nutno interpretovat v kontextu definice nových proměnných. Například to, že všechna řešení transformované rovnice konvergují k nule znamená, že všechna řešení původní rovnice konvergují k T_0. To, že řešení transformované rovnice klesne na poloviční hodnotu za čas $\ln 2$ znamená, že vzdálenost řešení původní rovnice od rovnovážného stavu se na polovinu zmenší za čas $\frac{1}{k} \ln 2$.

Poznámka (Nondimenzionalizace, rozměrová analýza)

Proces eliminace parametrů z modelu popsaného diferenciální rovnicí se nazývá nondimenzionalizace nebo rozměrová analýza modelu, protože eliminaci parametrů je vhodné provádět tak, aby výsledné nové veličiny vycházely bez fyzikálních jednotek. K tomu se provádí rozbor jednotek jednotlivých veličin. V jednoduchých případech však stačí primitivní postup popsaný v odstavcích výše a ukázaný na příkladu. V tomto příkladě veličina x nemá fyzikální jednotku, protože je součinem konstanty k (s jednotkou s^{-1}) a času t (s jednotkou s). Je možné ji považovat za *bezrozměrný čas*. Veličina y také nemá fyzikální jednotku, protože je podílem dvou teplot a je možné ji považovat za *bezrozměrnou teplotu*.

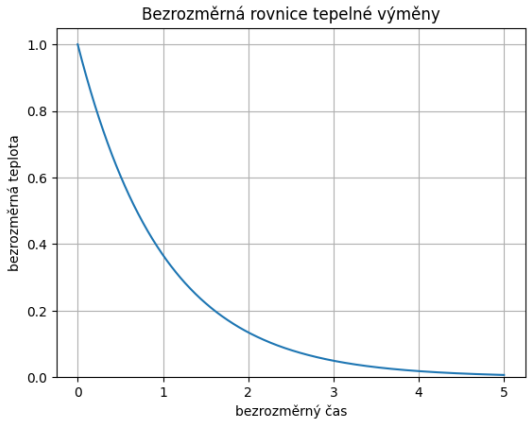
V úloze s ochlazováním tělesa bylo zavedení nových veličin přirozené. I u méně zřejmých úloh zkušenosti ukazují, že je vhodné volit transformaci tak, aby vznikly veličiny bezrozměrné, které nemají fyzikální jednotku.

Poznámka (Výhody transformace do bezrozměrných veličin.)

Obecné výhody transformace diferenciálních rovnic jsou následující.

- Po transformaci obsahuje rovnice v nových veličinách menší množství parametrů.
- Nové veličiny jsou bez fyzikální jednotky a tudíž vhodné pro numerické simulace, kdy se zpravidla o jednotky nestaráme.
- Nové veličiny zpravidla nabývají hodnot řádově srovnatelných s jedničkou. Nejedná se ani o tisíce ani o tisíciny.

Všechny tři uvedené skutečnosti vedou k tomu, že s transformovanými rovnicemi se lépe pracuje v numerických modelech.



Příklad (Bezrozměrná rovnice dynamické rovnováhy)

Rovnici

$$\frac{\mathrm{d}N}{\mathrm{d}t} = \frac{b}{D(N+\beta)} - a\frac{N^k}{S}.$$

rovnováhy počtu druhů na ostrově můžeme přepsat do tvaru

$$\beta\frac{\mathrm{d}\frac{N}{\beta}}{\mathrm{d}t} = \frac{b}{D\beta}\frac{1}{\frac{N}{\beta}+1} - \frac{a\beta^k}{S}\left(\frac{N}{\beta}\right)^k$$

a po vydělení faktorem stojícím před prvním zlomkem na pravé straně dostáváme

$$\frac{\beta^2 D}{b}\frac{\mathrm{d}\frac{N}{\beta}}{\mathrm{d}t} = \frac{1}{\frac{N}{\beta}+1} - \frac{a\beta^k}{S}\frac{\beta D}{b}\left(\frac{N}{\beta}\right)^k.$$

To ukazuje, že po zavedení bezrozměrného počtu druhů $n = \frac{N}{\beta}$ a bezrozměrného času $\tau = \frac{b}{\beta D}\frac{t}{\beta}$ má rovnice tvar

$$\frac{\mathrm{d}n}{\mathrm{d}\tau} = \frac{1}{n+1} - \alpha n^k$$

závislý na exponentu k a bezrozměrném parametru $\alpha = \frac{a\beta^k}{S}\frac{\beta D}{b}$.

KAPITOLA 4

Autonomní diferenciální rovnice

Co se dozvíte v tomto textu

V tomto textu se budeme věnovat diferenciálním rovnicím, kde rychlost vývoje sledované veličiny nezávisí explicitně na čase, ale jenom na veličině samotné a na parametrech prostředí. Naprostá většina modelů splňuje tyto požadavky. Důsledkem nezávislosti na čase je jistá jednoduchost při kvalitativním popisu chování řešení.

Důležitým reprezentantem autonomních diferenciálních rovnic je logistická rovnice, která bývá výchozím kamenem při studiu rostlinných a živočišných populací a označení parametrů z této rovnice písmeny r a K dalo dokonce název základním životním strategiím živočichů i rostlin.

Foto: Albatros (Diomedea exulans) v letu. Tento pták je představitelem K-stratégů. Autor JJ Harrison , https://wikimedia.org.

Diferenciální rovnice

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x)$$

s neznámou funkcí $x(t)$ proměnné t se nazývá *autonomní*, nebo též nezávislá na čase. Proměnná t se nazývá čas.

Rovnici je možné řešit analytickou cestou. My se nebudeme zaměřovat na hledání analytického tvaru obecného řešení, ale pokusíme se popsat chování řešení, aniž bychom tato řešení znali. Pokusíme se s co nejmenší námahou říct, jak se budou řešení chovat.

- Je-li $f(x_0) = 0$, je konstantní funkce $x(t) = x_0$ řešením rovnice. Protože derivace konstantní funkce je nula, vidíme, že řešením rovnice

  $$f(x) = 0$$

  obdržíme všechna konstantní řešení rovnice. Tato konstantní řešení se nazývají *stacionární body*.
- Stacionární body a jim odpovídající konstantní řešení představují rovnovážný stav. Často nás zajímá, jestli při vychýlení z tohoto rovnovážného stavu má systém tendenci se vrátit do původního stavu, nebo se od původního stavu dále odchylovat.
- Pokud se při malém vychýlení z rovnovážného stavu systém do tohoto stavu vrací, mluvíme o *stabilním stacionárním bodu*.
- Pokud se systém po malé výchylce do tohoto rovnovážného stavu nevrací, ale vyvíjí se k dalšímu stacionárnímu bodu nebo neohraničeně, mluvíme o *nestabilním stacionárním bodu.*

Následující věta umožní odlišit stabilní a nestabilní stacionární body. Protože v přírodě dochází k drobným perturbacím neustále, udává vlastně, které stacionární stavy jsou realizovatelné a můžeme je v přírodě pozorovat a které jsou prakticky nerealizovatelné.

Věta (Stabilita konstantních řešení)

Jestliže platí $f(x_0) = 0$, je konstantní funkce $x(t) = x_0$ konstantním řešením rovnice

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x).$$

Toto řešení je stabilní pokud funkce f v bodě x_0 klesá a nestabilní pokud funkce f v bodě x_0 roste.

Protože funkce f má ve stacionárním bodě nulovou funkční hodnotu, můžeme růst nebo pokles této funkce odhalit i ze znaménka. Pokud se znaménko funkce ve stacionárním bodě mění z kladného na záporné, potom funkce klesá. V opačném případě roste.

Pravá strana bývá často rozdílem dvou faktorů, které působí proti sobě v tom smyslu, že jeden definuje rychlost růstu a druhý rychlost poklesu sledované veličiny. V takovém případě se hodí následující důsledek předchozí věty.

Důsledek (Stabilita konstantních řešení rovnice s rozdílem na pravé straně)

Jestliže platí $f(x_0) = g(x_0)$, je konstantní funkce $x(t) = x_0$ konstantním řešením rovnice

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x) - g(x).$$

Toto řešení je stabilní pokud je funkce funkce $f(x) - g(x)$ kladná v levém okolí a záporná v pravém okolí bodu x_0. V případě opačných znamének je tento bod nestabilní.

4.1 Exponenciální růst

Nejjednodušším modelem růstu populace je model vyjadřující situaci, kdy rychlost růstu je úměrná velikosti populace. Takový růst je popsán diferenciální rovnicí

$$\frac{\mathrm{d}x}{\mathrm{d}t} = kx$$

a jeho analytickým řešením vyhovujícím podmínce $x(0) = x_0$ je funkce

$$x(t) = x_0 e^{kt}.$$

Model exponenciálního růstu je vhodný pro svou jednoduchost, ale nemůže popisovat dlouhodobý vývoj populace, protože při značné velikosti populace nedokáže daná lokalita populaci uživit. Model se nazývá Malthusův model a je použitelný pouze v krátkých časových intervalech a v situacích, kdy se neprojeví vnitrodruhová konkurence.

4.2 Von Bertalanffyho růst

Dalším modelem růstu je model, kdy je růst omezen a rychlost růstu je úměrná vzdálenosti vzdálenosti od horní hranice. Původně byl formulován pro růst jedinců, kteří se mohou dožívat maximální výšky L_∞. Podle tohoto modelu je výška L dána vztahem

$$\frac{\mathrm{d}L}{\mathrm{d}t} = k(L_\infty - L)$$

resp.

$$\frac{\mathrm{d}L}{\mathrm{d}t} = k_0 \left(1 - \frac{L}{L_\infty}\right),$$

kde $k_0 = \frac{k}{L_\infty}$. Analytickým řešením modelu s počáteční podmínkou $L(0) = L_0$ je funkce

$$L(t) = L_\infty + (L_0 - L_\infty)e^{-kt}.$$

Tento vztah ukazuje, že rozdíl délky a cílové délky se exponenciálně snižuje.

Kromě jedinců rostoucích rychlostí úměrnou množství chybějícímu do maximální velikosti tato rovnice popisuje i procesy, kdy změnu sledované veličiny způsobují dva faktory: růst konstantní rychlostí a pokles rychlostí úměrnou velikosti. Potom má totiž rovnice tvar

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha - \beta x$$

a po vytknutí konstanty α nebo β dospíváme (až na označení) k rovnicím výše. Tento růst například odpovídá sledování množství enzymů nebo bílkovin, které jsou produkovány konstantní rychlostí a degradují rychlostí úměrnou množství, viz [2] kapitola 1.4.

4.3 Logistický růst

Modelem pro růst populací v prostředí s nosnou kapacitou prostředí, který je všeobecně přijímán jako vhodný kompromis mezi přesností popisu a matematickou jednoduchostí je Verhulst-Pearlův model růstu, vycházející z předpokladu, že rychlost růstu populace v prostředí s omezenou nosnou kapacitou je úměrná velikosti této populace a volnému místu v životním prostředí, nejčastěji vyjádřenému procentem z nosné kapacity. Pokud nosná kapacita prostředí je K a specifická rychlost růstu bez započtení konkurence r, má model tvar

$$\frac{\mathrm{d}x}{\mathrm{d}t} = rx\left(1 - \frac{x}{K}\right)$$

při vyjádření rychlosti růstu a

$$\frac{1}{x}\frac{\mathrm{d}x}{\mathrm{d}t} = r\left(1 - \frac{x}{K}\right)$$

při vyjádření specifické rychlosti růstu. Pro x rovno nule je pravá strana rovnice se specifickou rychlostí růstu rovna r. Odpovídá to specifické rychlosti růstu populace, která roste v prostředí z nulové hodnoty. Proto se parametr r nazývá *invazní parametr*.

Rovnici je možno přepsat do tvaru

$$\frac{\mathrm{d}\frac{x}{K}}{\mathrm{d}(rt)} = \frac{x}{K}\left(1 - \frac{x}{K}\right)$$

a po zavedení bezrozměrné velikosti populace $y = \frac{x}{K}$ a bezrozměrného času $\tau = rt$ má rovnice tvar

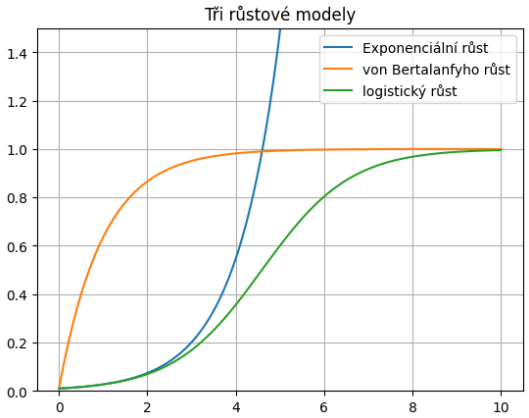
$$\frac{\mathrm{d}y}{\mathrm{d}\tau} = y(1-y).$$

To znamená, že bez újmy na obecnosti můžeme v logistické rovnici považovat nosnou kapacitu prostředí i inverzní parametr za jednotkové. Přesněji, vhodnou volbou jednotek můžeme docílit toho, že v uvedených jednotkách budou tyto konstanty numericky rovny jedné. Tohoto docílíme tak, že velikost populace se nebude měřit v počtech kusů, ale v násobcích nosné kapacity prostředí. Podobně, jednotka času se upraví tak, aby invazní parametr byl roven jedné. Tedy jednotkou času bude doba, za jak dlouho by populace dorostla do nosné kapacity prostředí, kdyby populace po celou dobu rostla konstantní rychlostí odpovídající rychlosti růstu od nulové hodnoty.

Poznámka (r-stratégové a K-stratégové)

Prostředí, ve kterém se populace nacházejí, je stále vystaveno změnám a výkyvům. To se odráží i na velikosti populace. Různé druhy živočichů se s těmito perturbacemi vyrovnávají různým způsobem. Některé druhy mají takřka stabilní stavy. Například populace albatrosů (*Diomedea exulans*) na ostrově Gough Island zůstávala téměř konstantní cca 4000 jedinců od roku 1889 přinejmenším do roku 1971 (viz [1] kapitola 5.3). Takové živočišné druhy nazýváme K-stratégy. Živočišné druhy, jejichž strategií je naopak se v příhodné době rychle rozmnožit, nazýváme r-stratégy a při modelování vývoje populace se tento faktor projevuje vysokým koeficientem r. Terminologie r-strategie a K-strategie je připisována MacArthurovi a Wilsonovi, autorům teorie ostrovní biogeografie.

Více o logistické rovnici viz například [3] kapitola 6.8, [14] kapitola 1.1 a [15] str. 182.



4.4 Autoregulace a rychlost syntézy proteinů

> Tento text je zpracován podle knihy [2] a pojednává o tom, jak zpětná vazba může ovlivnit produkci proteinu. Ukazuje několik strategií, jak může příroda zajistit, že v případě potřeby je protein syntetizován rychle a v dostatečném množství. Zaměříme se na matematickou stránku věci. Pěkná přednáška autora knihy a jednoho ze klíčových vědců v systémové biologii, moderní vědě, která se touto problematikou zajímá, je k dispozici na Youtube zde.

Uvažujme model syntézy proteinu, který degraduje rychlostí $s(x) = \beta x$, kde β je konstanta, tj. rychlostí úměrnou množství proteinu x. To je realistický a měřením potvrzený předpoklad. Kromě toho je protein ještě i syntetizován jiným procesem. Při dosažení rovnováhy mezi degradací a produkcí je jeho úroveň konstantní.

Schopnost řízení syntézy správných proteinů ve správnou dobu je klíčová pro všechny organismy. Pro některé jednodušší organismy je vzájemný řetězec na sebe navazujících reakcí do značné míry zmapován a popsán. Tato spleť reakcí je výsledkem dlouhodobé evoluce, kde náhodné genové mutace změnily chod dosud zavedených věcí. Tyto mutace buď přinesly svému nositeli výhodu a udržely se a rozšířily, nebo naopak přinesly nevýhodu a jejich nositel mutaci dále nešířil, protože nebyl tak úspěšný v reprodukci.

V této podkapitole si ukážeme, jak se dá různými strategiemi ovlivnit rychlost, s jakou je v buňce syntetizován protein. Nejedná se o jedinou záležitost, kterou v souvislosti s chemickými reakcemi v buňce studujeme matematickými prostředky. Dalšími problémy jsou například stabilita a robustnost stacionárních stavů. Přístupy jsou jak kvalitativní (může to takto vůbec fungovat?), tak kvantitativní (dostáváme při předpovědích vypočtených z modelů správné numerické hodnoty pro pozorované veličiny?) a v obou přístupech je dovednost modelování pomocí diferenciálních rovnic nezastupitelnou pomůckou.

4.4.1 Produkce konstantní rychlostí

Množství proteinu označme x a předpokládejme produkci konstantní rychlostí. Spolu s degradací z předchozího odstavce tedy máme model

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha - \beta x.$$

Nondimenzionalizace

Vhodnou volbou jednotek pro veličinu x je možné docílit toho, že platí $\alpha = \beta$. Opravdu, rovnici je možno přepsat na tvar

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha\left(1 - \frac{\beta x}{\alpha}\right)$$

a odsud po vynásobení faktorem $\frac{\beta}{\alpha}$

$$\frac{\mathrm{d}\frac{\beta x}{\alpha}}{\mathrm{d}t} = \beta\left(1 - \frac{\beta x}{\alpha}\right),$$

což v nových jednotkách $X = \frac{\beta x}{\alpha}$ vede na rovnici

$$\frac{\mathrm{d}X}{\mathrm{d}t} = \beta\,(1 - X)\,.$$

Tato volba znamená, že množství proteinu měříme v procentech rovnovážného stavu. Vhodnou volbou jednotky času je možné dosáhnout toho, že společná konstanta β je numericky rovna jedné. Tuto volbu však dělat nebudeme, protože chceme porovnávat časový průběh pro různé hodnoty konstanty β. Pro pohodlí se navíc vrátíme k původnímu označení malým písmenem, tj. budeme studovat rovnici

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \beta\,(1 - x)\,.$$

Toto je rovnice

$$\frac{\mathrm{d}(x-1)}{\mathrm{d}t} = -\beta\,(x - 1)\,,$$

tj. rovnice popisující rozklad rychlostí úměrnou množství pro veličinu $x - 1$ a řešením je tedy

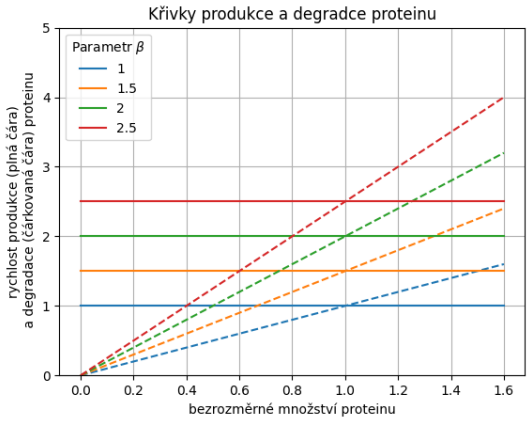
$$x(t) = 1 - (1 - x_0)e^{-\beta t}.$$

Přestože máme k dispozici analytické řešení, vyřešíme si úlohu numericky, abychom mohli poté volit různé rychlosti produkce.

Růstové křivky

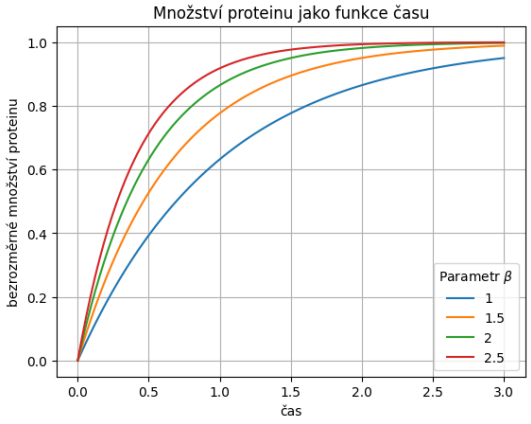
Nejprve nakreslíme křivky produkce a degradace do jednoho obrázku, abychom dokázali lokalizovat stacionární body.

- Křivky pro stejné hodnoty parametru jsou stejnou barvou.
- Průsečík křivek stejné barvy je stacionární bod systému při nastavení parametrů daném příslušnou barvou. Všimněte si, že všechny tyto průsečíky jsou nad sebou a každý v jiné výšce. Pro všechny parametry nastává stacionární stav pro stejné množství proteinu $x = 1$, ale rychlosti produkce a degradace se v tomto stacionárním stavu liší.
- Všechna nastavení parametrů vykazují převažující růst pro malé a převažující degradaci pro velké hodnoty x. Stacionární body jsou tedy stabilní.



Časový vývoj řešení

Průběh řešení pro jednotlivé hodnoty parametru u řešení vycházejícího z nulové počáteční podmínky je na obrázku. V praxi nás zajímá, jak rychle bude dosaženo určitého množství proteinu. Tedy vybereme si nějakou hladinu, například 0.6, a sledujeme, která křivka této hladiny dosáhne jako první a která jako poslední.



Rychlá produkce je „zaplacena“ rychlou degradací

Je patrné, že nejrychleji hladina enzymu roste pro nejvyšší hodnotu parametru. Vodorovnou čáru ve výšce 0.6 protne jako první křivka odpovídající nejvyšší hodnotě parametru β. Být první je neskutečně důležité, někdy přímo otázka života a smrti. Taková bakterie, která umí rychle produkovat enzym, například umí rychle uniknout z prostředí s nepříznivými životními podmínkami (rychle sestaví proteiny, které vytvoří jakýsi molekulární motor a ten buňku posune na jiné místo) nebo umí rychle začít ukládat v přítomnosti glukózy energii pro budoucí využití. Na druhou stranu, popsaným mechanismem je rychlost je dosažena nejenom rychlou produkcí, ale i rych-

lou degradací proteinu. Tedy protein se rychle produkuje a rychle rozpadá. Už z laického hlediska se takový postup nezdá být postupem optimálním pro přežití. Je to jako bychom u auta chtěli mít možnost okamžitě zrychlit a za tímto účelem udržovali vysoké otáčky motoru, ale abychom jeli rozumnou rychlostí tak bychom brzdili. Tedy bychom současně šlapali na plyn i na brzdu. To rozumný člověk nedělá.

Proteiny ve skutečnosti nejsou rychle odbourávány. Popsaný mechanismus šlapání na brzdu a plyn současně v buněčném světě ve skutečnosti není použit. Životnost proteinu není řádově odlišná od životnosti buňky. Příroda a evoluce by takové plýtvání nepřipustily a namísto toho se k problému rychlé produkce proteinu staví jinak. Použitím zpětné vazby, což je předmětem následující kapitoly.

Bezrozměrná formulace

Zkusíme ještě přepsat rovnici s konstantní produkcí do bezrozměrných jednotek. Rovnici

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \beta(1-x)$$

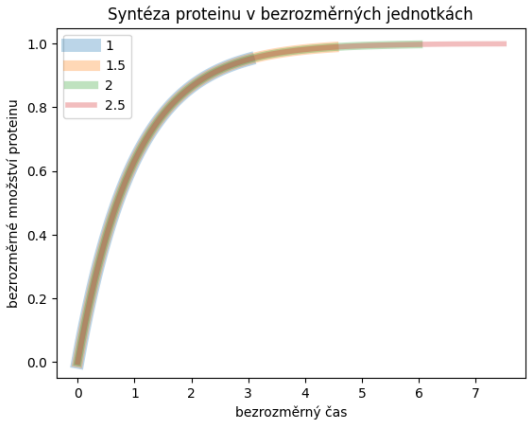
je možno přepsat do tvaru

$$\frac{\mathrm{d}x}{\mathrm{d}(\beta t)} = 1-x$$

a pro bezrozměrný čas $\tau = \beta t$ se rovnice redukuje na

$$\frac{\mathrm{d}x}{\mathrm{d}\tau} = 1-x.$$

Rovnice je nezávislá na parametrech a po převodu na bezrozměrný čas (tj. po vynásobení času parametrem β) všechna řešení budou splývat. Protože v grafu se budou překrývat, vykreslíme je různě širokými poloprůhlednými čarami.



Vidíme názorně výhodu bezrozměrných jednotek. Není nutné řešit rovnici pro různé hodnoty parametru, ale všechna řešení jsou až na transformaci stejná. Jinými slovy vyjádřeno, v modelu

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha - \beta x$$

produkce konstantní rychlostí je možno bez újmy na obecnosti položit oba parametry α a β rovny jedné.

4.4.2 Produkce se zápornou zpětnou vazbou

Předpokládejme produkci proteinu rychlostí související s množstvím proteinu. Jedná se tedy o model

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x) - \beta x,$$

kde $f(x)$ je rychlost produkce. Budeme porovnávat vliv funkce $f(x)$ na rychlost syntézy.

Vhodný tvar funkce $f(x)$ může být například rozdíl konstanty a třetí mocniny, tj.

$$f(x) = a - bx^3.$$

Čím vyšší hodnota koeficientu α, tím je počáteční produkce rychlejší a dříve je dosaženo potřebného množství enzymu. Toto se děje při konstantní zpětné vazbě a konstantním stacionárním stavu (všechny rovnice mají člen popisující degradaci ve tvaru $-x$ a stacionární stav $x = 1$).

Nondimenzionalizace

Vhodnou volbou jednotek pro veličinu x dosáhneme toho, že stacionárním bodem bude $x = 1$ (jinými slovy, množství enzymu budeme měřit v procentech rovnovážného stavu). Protože budeme uvažovat stále stejnou rychlost degradace a sledovat jenom vliv funkce $f(x)$, můžeme jednotky času zvolit tak, že $\beta = 1$. Tj. studujeme rovnici

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x) - x,$$

kde $f(1) = 1$. Aby byla navíc splněna podmínka $f(1) = 1$, budeme uvažovat funkci

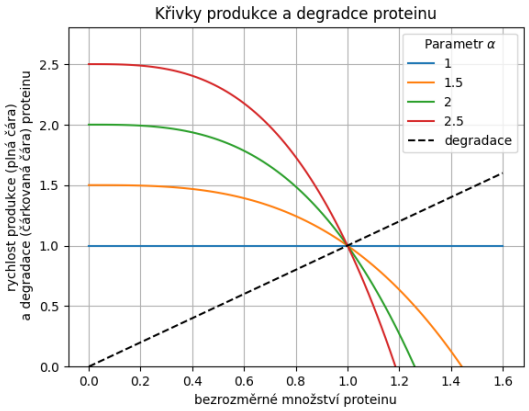
$$f(x) = \alpha - (\alpha - 1)x^3, \quad \alpha \geq 1.$$

Křivky růstu a degradace

Funkce

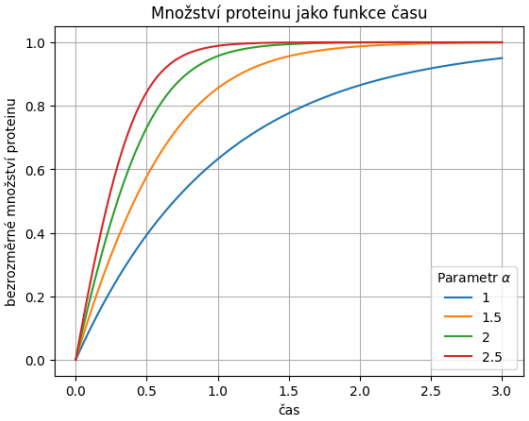
$$f(x) = \alpha - (\alpha - 1)x^3, \quad \alpha \geq 1$$

je klesající funkce. Rychlost produkce tedy ubývá s množstvím. Toto se nazývá záporná zpětná vazba. Parametr α reguluje sílu této zpětné vazby, čím je vyšší, tím je zpětná vazba výraznější. Vyšší α charakterizuje vyšší hodnotu průsečíku na svislé ose (rychlejší produkci při nulové hladině proteinu) a rychlejší pokles při rostoucí koncentraci. Pěkně to je vidět na následujícím obrázku.



Časový průběh řešení modelu

Zkusíme si namodelovat průběh řešení modelu pro různé hodnoty parametru.



Čím vyšší hodnota koeficientu α, tím je počáteční produkce rychlejší a dříve je dosaženo potřebného množství enzymu. Toto se děje při konstantní zpětné vazbě a konstantním stacionárním stavu (všechny rovnice mají člen popisující degradaci ve tvaru $-x$ a stacionární stav $x = 1$).

Negativní zpětná vazba urychluje syntézu

Vidíme, že negativní zpětná vazba příznivě ovlivňuje rychlost syntézy proteinu. Tím je buňce umožněno zareagovat rychle na vnější podmínky, aniž by za to platila cenou rychlé degradace, jako bez zpětné vazby (současně přidávání plynu a brzdění). Taková zpětná vazba může být zprostředkována například tím, že protein vstupuje do jiných chemických reakcí.

Příroda zařadila i do jednoduchých organismů typu buňka takových zpětných vazeb obrovské množství a dlouholetou evolucí vyladila hodnoty parametrů tak, aby potřebné proteiny vznikaly přesně v době, kdy jsou potřeba a aby dostatečně rychle vznikly v potřebné koncentraci. To je zařízeno mutacemi v genech. Chybou v přepise genů například může vzniknout zpětná vazba, která do té doby neexistovala. Pokud se projeví jako užitečná, dává svému nositeli evoluční výhodu a šíří se dál. Podobně se během milionů let vyladí parametry této zpětné vazby, například hodnota parametru α. Těmito problémy zabývá věda nazývaná systémová biologie. Tato věda prodělala bouřlivý rozvoj v letech okolo roku 2000, kdy byly vyvinuty účinné postupy pro získávání potřebných dat o genech a proteinech a tento rozvoj pokračuje dodnes.

Poznatky, na kterých je založen tento model, jsou potvrzeny experimentálně. Používá se například vnesení genu způsobujícího světélkování do genetické informace bakterií.

KAPITOLA 5

Bifurkace

Co se dozvíte v tomto textu

V tomto textu se budeme věnovat vlivu parametrů na chování řešení diferenciální rovnice. Zvláštní pozornost bude věnována situacím, kdy může dojít ke jiným než plynulým změnám fázového portrétu, například zániku nebo ztrátě stability stacionárního bodu.

Techniky sledující závislost řešení na parametrech jsou zásadním při managementu populací, zejména při návrhu trvale udržitelného lovu. Historicky byly nejprve rozpracovány pro rybolov, aby byl přehled o důsledcích lovu na život v oceánech.

Kromě základní metodiky a aplikací na lov populace se seznámíme i s problematikou redukce populace škůdce působením predátorů a následky, které mohou nastat při překročení hranice, při které se mění struktura stacionárních bodů modelu, jejich počet a stabilita.

Foto: S bifurkacemi se setkáváme v situacích, kdy se podstatně mění chování systému. V praxi to může znamenat například zhroucení ekosystému, jak se několikrát stalo v případě nadměrného rybolovu. Na snímku rybářské sítě pro rybolov malého rozsahu. Anoop Negi, https://www.flickr.com/photos/ezee123/117560929

Jak jsme viděli v předešlých modelech, v praxi naše modely obsahují parametry. Tyto parametry ovlivňují chování modelu v několika ohledech.

Zpravidla se stává, že malé změny parametrů mají malý vliv na chování řešení. Ovlivní pouze polohu stacionárních stavů a rychlost konvergence řešení do těchto bodů, ale při malé změně parametrů se zpravidla stacionární body posunují o malé hodnoty a jejich počet i jejich stabilita zůstávají zachovávány. V zásadě se pro všechny hodnoty parametru řešení chovají pořád stejně. To jsme viděli například u diferenciální rovnice ochlazování

$$\frac{\mathrm{d}T}{\mathrm{d}t} = -k(T - T_\infty).$$

Řešení ukazuje, že těleso se ochladí na teplotu okolí pro libovolnou kombinaci konstanty úměrnosti k a teploty okolí T_∞. Toto plyne i z faktu, že v bezrozměrném tvaru rovnice ochlazování neobsahuje žádný parametr.

Někdy se však stane, že změna parametru při překročení nějaké hodnoty vyvolá velkou změnu chování systému. Například zanikne nebo vznikne nový stacionární bod, nebo se změní jeho stabilita. V takovém případě říkáme, že rovnice má v daném bodě *bifurkaci*. Příslušná hodnota parametru se nazývá *bifurkační hodnota*.

5.1 Bifurkace v logistické diskrétní rovnici

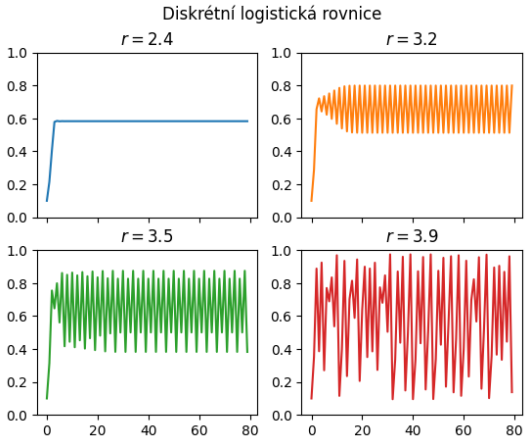
Nejjednodušší je pozorovat bifurkaci na diskrétní obdobě logistické rovnice, na rovnici

$$x_{k+1} = rx_k(1 - x_k)$$

Rovnice má pro malé r stacionární bod a řešení konverguje podobně jako u spojité logistické rovnice k nosné kapacitě prostředí. Pokud hodnotu r zvyšujeme, stacionární bod se posunuje, ale jinak se nic neděje dokud nepřekročíme jistý kritický bod. Za hodnotou $r = 3$ už

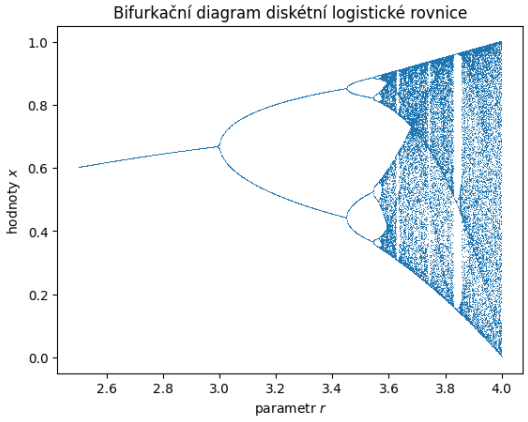
řešení přestává konvergovat k jedné hodnotě, ale přeskakuje mezi dvěma hodnotami, má cyklus s periodou 2. Při dalším zvyšování se jenom posunují body, mezi kterými řešení osciluje, dokud se nedostaneme k další kritické hodnotě. Pro r mezi přibližně 3.44949 a 3.54409 začne řešení periodicky přeskakovat mezi čtyřmi hodnotami a jedná se tedy o cyklus s periodou 4.

Pro hodnoty parametru r použité v obrázcích se řešení buď ustálí na konstantní hodnotě, nebo vznikne cyklus s periodou dva nebo cyklus s periodou čtyři. V posledním případě dokonce vznikne zdánlivě chaotické chování, které je ve skutečnosti cyklem s s dlouhou periodou.



Vidíme, že chování rovnice je velice pestré. Následující diagram je takzvaný bifurkační diagram této rovnice. Na vodorovné ose je hodnota parametru. Pokud parametru odpovídá jedna hodnota na svislé ose, řešení se ustálí na dané hodnotě. Pokud více, řešení mezi těmito hodnotami přeskakuje v cyklu. V místě, kde se křivka rozdvojuje se mění počet hodnot, mezi kterými řešení přeskakuje a mění se tedy perioda cyklu, ke kterému řešení konverguje. V takovém bodě má rovnice bifurkaci.

Čím více průsečíků má graf se svislou přímkou, tím větší je perioda řešení a tím chaotičtější je systém, protože se střídají řádově desítky či stovky hodnot.



5.2 Logistický růst a dvě strategie lovu

Předpokládejme, že populace vyvíjející se podle logistické rovnice a populace je vystavena lovu, odchytu či těžbě. Naším cílem je zjistit, jaké bude mít tento lov důsledky na stav a vývoj populace.

5.2.1 Konstantní intenzita lovu

Předpokládejme, že lov je prováděn tak, že rychlost, s jakou odebíráme jedince z populace, je konstantní. Označme tuto rychlost h. Model vývoje populace má tedy tvar

$$\frac{\mathrm{d}x}{\mathrm{d}t} = rx\left(1 - \frac{x}{K}\right) - h.$$

Nejprve zredukujeme počet parametrů. Rovnici je možno přepsat do tvaru

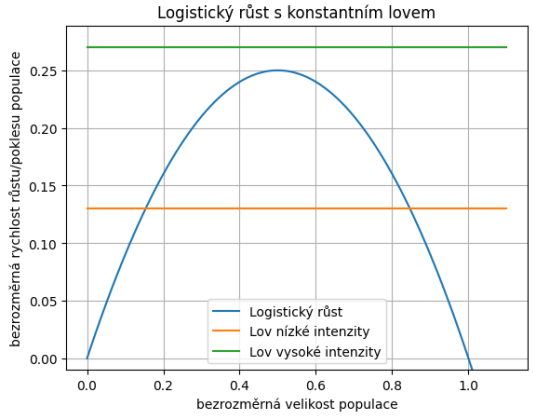
$$\frac{\mathrm{d}\frac{x}{K}}{\mathrm{d}(rt)} = \frac{x}{K}\left(1 - \frac{x}{K}\right) - \frac{h}{Kr},$$

a po zavedení bezrozměrné velikosti populace $y = \frac{x}{K}$, bezrozměrného času $\tau = rt$ a bezrozměrného lovu $\alpha = \frac{h}{Kr}$ má bezrozměrná rovnice lovu konstantní intenzity tvar

$$\frac{\mathrm{d}y}{\mathrm{d}\tau} = y\,(1 - y) - \alpha.$$

Tip: Téhož tvaru bezrozměrné rovnice dosáhneme, pokud budeme velikost populace měřit v násobcích nosné kapacity prostředí a čas v takových jednotkách, aby byl bez lovu invazní parametr roven jedné. Tedy aby bez lovu malá populace rostla rychlostí odpovídající dosažení nosné kapacity prostředí za jednotku času. V praxi se často tato úvaha odbude tvrzením „invazní parametr a nosnou kapacitu prostředí můžeme bez újmy na obecnosti položit rovny jedné“.

Funkce stojící na pravé straně diferenciální rovnice v rozdílu si vykreslíme.

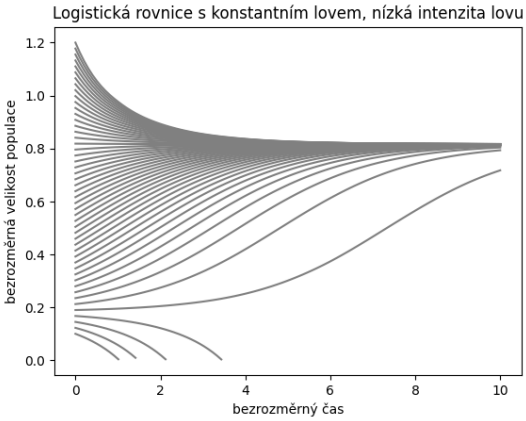


Grafem funkcí, jejichž rozdíl figuruje na pravé straně rovnice, jsou parabola řídící růst a vodorovná přímka vyjadřující lov. Změnou intenzity lovu se vodorovná přímka posunuje nahoru (zvýšení), či dolů (snížení). Pro lov nízkou intenzitou má přímka s parabolou dva průsečíky a ty se posunují, jak se vodorovná přímka posunuje nahoru nebo dolů. Pro lov s vysokou intenzitou průsečíky zaniknou.

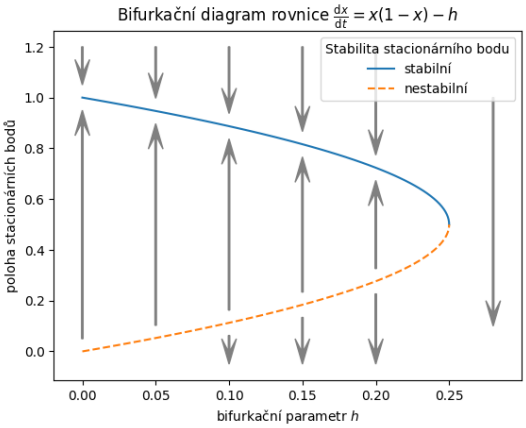
Pro vysoký lov tedy je rozdíl růstu a poklesu záporný a velikost populace klesá v čase. Tato rychlost poklesu je minimálně tak velká, jako vzdálenost vodorovné přímky od vrcholu paraboly a proto není možné očekávat, že by se pokles zastavil tak, že řešení konverguje k vodorovné asymptotě. V této situaci populace nepřežije a lov ji zlikviduje.

Pro lov nízké intenzity mají křivky růstu a poklesu dva průsečíky. Nalevo od prvního průsečíku převažuje pokles nad růstem a populace vymře. Napravo od druhého průsečíku také převažuje pokles nad přírůstkem, populace vymírá, ale toto vymírání se zastaví ve stacionárním bodě odpovídajícím tomuto průsečíku. Mezi stacionárními body převažuje růst nad lovem a populace roste.

Model ukazuje, že pro vysoké hodnoty lovu populace zanikne a pro rozumné hodnoty lovu přežívá. Toto přežívání je podmíněno tím, že velikost populace na začátku vývoje není pod nestabilním stacionárním bodem a ani pod něj náhodnými fluktuacemi neklesne.



Bifurkační diagram je na obrázku. Pro různé hodnoty parametru je na něm možné najít polohu stacionárních bodů a identifikovat jejich stabilitu. Pro hodnotu $h = 0.25$ nastává bifurkace, kdy při zvyšování lovu splývají a zanikají oba stacionární body.

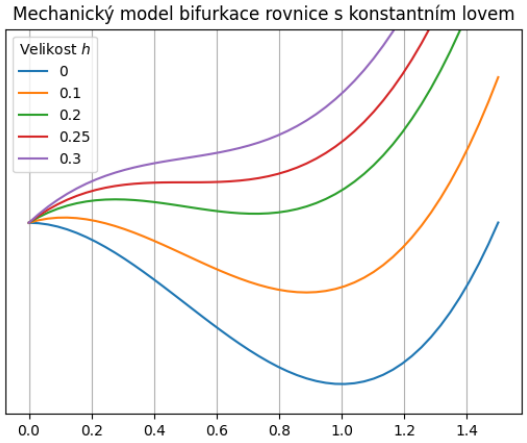


Pro přežívání populace je nutné hlídat, aby populace nikdy neklesla pod kritickou hodnotu vedoucí k vymření populace. Proto je dobré udržovat hodnotu lovu na takové úrovni, aby vzdálenost mezi stabilním a nestabilním stacionárním bodem byla dostatečně velká, aby náhodné perturbace, ke kterým může kdykoliv dojít, nesnížily velikost populace pod nestabilní práh. V takovém případě při poklesu populace není nutné měnit strategii lovu, populace drobné fluktuace vyrovná sama svými přirozenými růstovými mechanismy.

V praxi je nutno buď uvažovat co největší h, abychom měli co největší užitek, sledovat, zda vlivem výkyvů velikost populace neklesá pod kritickou hodnotu a pokud ano tak okamžitě měnit rychlost lovu nebo další parametry systému (podpořit rozmnožování a pod.). Další alternativou je lovit menší rychlostí h, což nese nižší užitek z lovu, ale ponechává populaci možnost, aby se sama vyrovnala s případnými výkyvy.

Poznamenejme, že v uvedeném modelu má smysl i předpoklad, že koeficient h je záporný. V takovém případě nemluvíme o záporném lovu, ale příslušný člen interpretujeme jako rychlost imigrace, se kterou uvažovaný druh proniká do dané lokality.

Následující model ukazuje mechanickou představu bifurkace pomocí analogie s kuličkodráhou. Je vykreslen potenciál (záporně vzatý integrál) pravé strany. Ten je možno chápat jako potenciální energii a chování systému je možné přirovnat k chování kuličky pohybující se po kuličkodráze ve stejném tvaru jako je tento potenciál. Pro malé hodnoty lovu má potenciál lokální minimum a v tomto minimu existuje stabilní stacionární stav. S rostoucím lovem je minimum stále méně výrazné a i malá perutbace může kuličku z dolíku vyhodit a kulička se skutálí pryč. Pro kritický lov $h = 0.25$ už lokální minimum zaniká. Pro ještě intenzivnější lov se celá křivka svažuje doleva a kulička se odkutálí pryč.



5.2.2 Lov úměrný velikosti populace

Uvažujme strategii lovu spočívající v tom, že rychlost, s jakou lovem snižujeme velikost populace, je úměrná celkové velikosti populace. Malou populaci lovíme málo, větší populaci více. To by odpovídalo například filtrování, kdy odstraňujeme bakterie z živného roztoku.

Matematicky vyjádřeno, populace se vyvíjí podle modelu

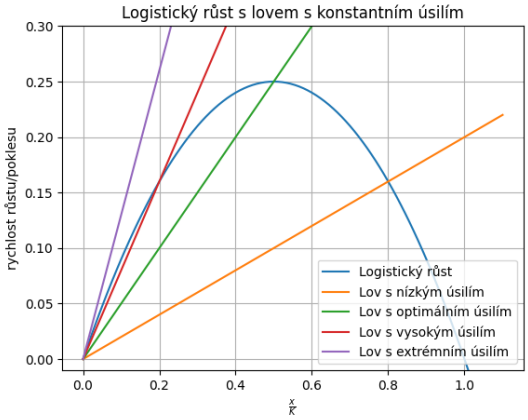
$$\frac{\mathrm{d}x}{\mathrm{d}t} = rx\left(1 - \frac{x}{K}\right) - qx$$

kde r, q, K jsou kladné konstanty. První člen na pravé straně je parabola známá z logistické rovnice, druhý člen odpovídá přímce procházející počátkem.

V bezrozměrném tvaru má rovnice podobu

$$\frac{\mathrm{d}y}{\mathrm{d}\tau} = y\,(1 - y) - \alpha y,$$

kde $y = \frac{x}{K}$, $\tau = rt$ a $\alpha = \frac{q}{r}$. Tento tvar vyjadřuje, že v rovnici je možné bez újmy na obecnosti položit hodnoty invazního parametru a nosné kapacity prostředí rovny jedné.



Vidíme, že tato strategie lovu má oproti konstantnímu lovu velkou výhodu: nemá nestabilní stacionární bod. (S výjimkou extrémně vysoké intenzity lovu, kdy je směrnice přímky charakterizující lov v bezrozměrných veličinách větší než jedna.)

Pro praktické provádění této strategie je však nutno znát v každém okamžiku aktuální velikost populace, což může být obtížně zjistitelné, nebo takřka nemožné. V takovém případě je vhodnější použít strategii konstantního lovu. V případě, kdy velikost úlovku je úměrná intenzitě lovu (např. vynaloženému úsilí, kvalitě loveckých pomůcek a pod.), je naopak snadné realizovat strategii s proměnnou rychlostí, protože stačí lovit se stále stejnou intenzitou q.

V praxi se snažíme užitek z lovu maximalizovat a tedy se snažíme o to, aby součin qx byl v rovnovážném stavu co největší. To znamená, že přímka charakterizující lov prochází vrcholem paraboly. Zvýšení úsilí vede ke strmější přímce, která protíná parabolu níže a celkový užitek z lovu se tedy snižuje.

Kromě toho je pro *ekonomicky* nejvýhodnější strategii lovu nutno uvažovat, že většího úlovku dosáhneme jenom za použití většího úsilí. Na toto zvýšení úsilí je nutno vynaložit větší ekonomické náklady, které snižují celkový zisk. Je tedy nutno hledat určitou rovnováhu – chceme odlovit co nejvíce jedinců, abychom měli co největší zisk, od určité hranice je však již lov drahý a nevyplácí se. Tato problematika, maximalizace ekonomického profitu, je podrobně rozpracována v odborné literatuře, zejména na příkladech z rybolovu.

5.3 Populace pod predačním tlakem

Následující model má velký historický význam. Umožnil objasnit, proč v kanadských lesích dochází periodicky k přemnožování obaleče smrkového. Toto je motýl, jehož housenky v době přemnožení dokáží zničit obrovské plochy lesa. Na monitorování situace používala kanadská lesní správa model (Holling, Jones, Clark) obsahující 30 tisíc proměnných, které sledovaly prostorové rozložení populace a její vývoj v čase. Model však nedokázal odpovědět na otázku, proč dochází k periodickému přemnožení. Pokud pochopíme tento proces, budeme schopni mu účinně bránit.

Na vysvětlení uvedeného problému sestavili Ludwig, Jones a Holling model složený ze tří diferenciálních rovnic popisujících dynamiku lesa, housenek obaleče a jejich predátorů, ptáků. My si zde představíme zjednodušení, kdy se budeme věnovat jenom nejrychleji se měnící populaci, populaci housenek obaleče. Tato populace bude pod tlakem predátorů, ptáků a malých savců, kteří se housenkami živí.

5.3.1 Matematický model

Předpokládejme, že populace predátorů je stabilizovaná a že přežití predátorů není závislé na velikosti populace obaleče. To nastane zejména v případech, kdy má predátor v prostředí i alternativní zdroje potravy a kdy dynamika jeho populace je pomalejší než dynamika populace obaleče.

Situaci budeme modelovat diferenciální rovnicí

$$\frac{\mathrm{d}N}{\mathrm{d}t} = r\left(1 - \frac{N}{K}\right)N - V(N),$$

kde první člen na pravé straně rovnice odpovídá logistickému růstu obaleče a druhý člen na pravé straně charakterizuje, jak je vývoj populace zpomalován působením predátorů. Působení predátorů je charakterizováno následujícími znaky.

- Bez přítomnosti kořisti predátoři nic neuloví, tj. $V(0) = 0$.
- Je-li více kořisti, predátoři jí neuloví méně, tj. V je neklesající funkce
- Predátoři nemohou ulovit neomezeně mnoho kořisti, ale loví pouze do jisté hladiny nasycení, funkce V je proto ohraničená.
- Je-li populace kořisti malá, predátoři ji téměř neloví, ale dávají přednost dostupnějším zdrojům potravy, funkce V tedy zpočátku roste pomalu.

Ludwig, Jones a Holling navrhli použít pro funkci V trofickou funkci typu Holling II ve tvaru

$$V(N) = S\frac{N^2}{N^2 + b^2},$$

kde S a b jsou kladné konstanty. Uvažovanou úlohu poté modeluje rovnice

$$\frac{\mathrm{d}N}{\mathrm{d}t} = r\left(1 - \frac{N}{K}\right)N - S\frac{N^2}{N^2 + b^2}.$$

5.3.2 Nondimenzionalizace

Abychom mohli kvalitativně prozkoumat vlastnosti řešení, pokusíme se zavést bezrozměrné veličiny a tím poněkud zjednodušit pravou stranu rovnice. Budeme se snažit především eliminovat parametry z trofické funkce, protože tento člen na pravé straně rovnice je složitější než člen z rovnice logistického růstu.

Zavedeme novou závisle proměnnou x substitucí $N = bx$. Biologicky to znamená, že velikost populace budeme měřit v nových jednotkách, které jsou b-násobkem původních jednotek. Po této substituci má rovnice tvar

$$b\frac{\mathrm{d}x}{\mathrm{d}t} = rb\left(1 - \frac{bx}{K}\right)x - S\frac{x^2}{x^2 + 1}.$$

Tuto rovnici je možno dále přepsat do tvaru

$$\frac{b}{S}\frac{\mathrm{d}x}{\mathrm{d}t} = \frac{rb}{S}\left(1 - \frac{x}{\frac{K}{b}}\right)x - \frac{x^2}{x^2 + 1}.$$

Změníme-li dále jednotky měření času substitucí $\tau = \frac{tS}{b}$ a zavedeme nové parametry $\alpha = \frac{rb}{S}$, $\beta = \frac{K}{b}$, má rovnice pro vývoj populace tvar

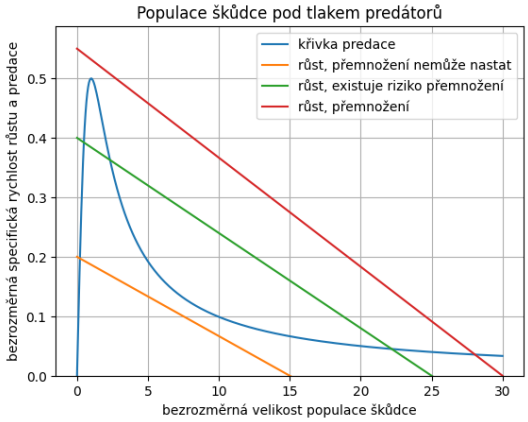
$$\frac{\mathrm{d}x}{\mathrm{d}\tau} = \alpha\left(1 - \frac{x}{\beta}\right)x - \frac{x^2}{x^2 + 1}.$$

Jinými slovy, bez újmy na obecnosti můžeme položit hodnoty S a b rovny jedné. Jedná se tedy o klasickou rovnici pro vývoj jednodruhové populace se specifickou mírou růstu

$$\mu(x) = \alpha\left(1 - \frac{x}{\beta}\right) - \frac{x}{x^2 + 1}.$$

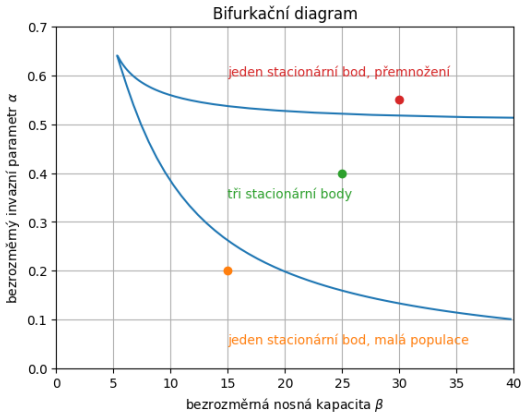
5.3.3 Růstové křivky

Funkci $\frac{x}{x^2+1}$ a přímku $\alpha\left(1 - \frac{x}{\beta}\right)$ vykreslíme do jednoho grafu. V závislosti na hodnotách parametrů α a β může vzájemná poloha dopadnout trojím způsobem. Buď se křivky protnou jenom jednou v bodě, kde je hodnota populace malá. Toto nastane pro malou nosnou kapacitu a malý invazní parametr. V takovém případě je populace housenek na nízké úrovni.



5.3.4 Bifurkační diagram

V prostoru parametrů můžeme dokonce najít křivky oddělující existenci jednoho stacionárního stavu a tří. Vznikne bifurkační diagram ukazující, pro které hodnoty parametrů jsou možné tři stacionární stavy (dva stabilní a jeden nestabilní) a pro které hodnoty existuje jenom jeden. Tečky v diagramu odpovídají přímkám, které jsou v zachyceny na předchozím obrázku. Barvy si odpovídají, červený bod charakterizuje červenou přímku a podobně.



5.3.5 Závěr

Model vysvětluje, proč dochází k periodickému přemnožování. Jak les roste, poskytuje větší nosnou kapacitu prostředí pro obaleče a lepší podmínky pro jeho množení. Tím se přímky v diagramu srovnávajícím produkci a predaci dostávají doprava nahoru a z původně jednoho nízkého stacionárního stavu (oranžová přímka) máme tři stacionární stavy s možností přemnožení a malým prahem mezi prvním a druhým stacionárním bodem (zelená přímka) a nakonec jeden stacionární stav znamenající přemnožení. Jakmile k tomuto přemnožení dojde, škůdce zničí les i své životní prostředí a les začíná růst od začátku. Tyto cykly pozorujeme v provincii Nový Brunšvik, protože v jiných provinciích dochází k jiným katastrofám omezujícím růst lesa, například lesní požáry ([8]).

Strategií ochrany lesa je vhodnými opatřeními nedovolit posun parametrů do oblasti, kde je jeden stacionární bod s vysokou hodnotou populace. V takových podmínkách totiž je nízká úroveň populace daleko od stabilního stavu a její udržování je možné pouze za cenu neustálých zásahů a vysokých investic. Po jakémkoliv jednorázovém zredukování populace se rychle obnovuje rovnováha odpovídající vysokým hodnotám. Pro účinný zásah je nutné vhodnými opatřeními zajistit, aby se parametry systému přesunuly do oblasti se třemi stacionárními body a poté dalším zásahem jednorázově zredukovat populaci. Jakmile populaci jednou zredukujeme pod hodnotu odpovídající nestabilními stacionárnímu bodu, bude konvergovat ke stabilnímu stacionárnímu bodu odpovídajícímu malé populaci a obaleč tedy nebude přemnožen.

KAPITOLA 6

Lineární algebra, matice

Co se dozvíte v tomto textu

V následujících odstavcích se budeme věnovat problematice mnohorozměrných veličin. Ty se používají například ke sledování populace rozdělené do několika věkových nebo vývojových tříd a umožňují formulovat modely zohledňující tuto strukturu.

Díky tomu je možno například identifikovat účinnou strategii ochrany živočišných druhů. Analýza pomocí prostředků lineární algebry umožní rozhodnout, zda je pro přežití populace nějakého druhu důležitější přežití dospělých jedinců podílejících se na reprodukci, nebo produkce mláďat. Umožní rozhodnout, zda pro šetrnou těžbu lesa je výhodnější kácení mladých nebo starých stromů. Umožní nastavit lov, který buď neohrozí stabilitu lovené populaci, nebo naopak (při eliminaci škůdců) dokáže zasáhnout populaci na nejcitlivějším místě.

Foto: Tuleň kuželozubý. Na modelu tohoto tuleně si ukážeme možnosti modelování populace s věkovou strukturou. Autor George Hodan, https://www.publicdomainpictures.net/.

Lineární algebra je odvětví matematiky, zabývající se vektory a obecně mnohorozměrnými veličinami.

6.1 Vektory

Vektorem rozumíme uspořádanou n-tici objektů, pro které má smysl operace sčítání a násobení číslem. Počet komponent v této n-tici se nazývá dimenze vektoru. Tyto komponenty jsou zpravidla čísla nebo skalární funkce. Aby se s vektory dalo rozumně pracovat, musí na nich být definovány matematické operace, které tvoří vhodnou matematickou strukturu. Například operace musí mít neutrální prvek a každý vektor musí mít opačný prvek.

Definice (Vektory, vektorový prostor)

Množinu V uspořádaných n-tic $(a_1, a_2, \dots, a_n)$ s operacemi sčítání a násobení reálným číslem definovanými

$$(a_1, a_2, \dots, a_n) + (b_1, b_2, \dots, b_n) = (a_1 + b_1, a_2 + b_2, \dots, a_n + b_n)$$
$$c \cdot (a_1, a_2, \dots, a_n) = (c \cdot a_1, c \cdot a_2, \dots, c \cdot a_n)$$

pro všechna $c \in \mathbb{R}$ a $(a_1, a_2, \dots, a_n), (b_1, b_2, \dots, b_n) \in V$ nazýváme *vektorovým prostorem*. Prvky tohoto prostoru nazýváme *vektory*. Prvky $a_1, \dots, a_n$ nazýváme *složky vektoru* $(a_1, a_2, \dots, a_n)$. Číslo n nazýváme *dimenze prostoru* V.

Vektorový prostor, jehož komponenty jsou uspořádané n-tice reálných čísel označujeme $\mathbb{R}^n$.

Často pracujeme se sloupcovými vektory. Zápis je potom přehlednější.

6.2 Lineární kombinace

Definice (Lineární kombinace)

Nechť $\vec{u}_1$, $\vec{u}_2$, ... $\vec{u}_k$ je konečná posloupnost vektorů z vektorového prostoru V. Vektor $\vec{u}$, pro který platí

$$\vec{u} = t_1\vec{u}_1 + t_2\vec{u}_2 + \cdots + t_k\vec{u}_k,$$

kde t_1, t_2, ..., t_k jsou nějaká reálná čísla, se nazývá *lineární kombinace* vektorů $\vec{u}_1$, $\vec{u}_2$, ..., $\vec{u}_k$. Čísla t_1, t_2, ..., t_k nazýváme *koeficienty lineární kombinace*.

Definice (Lineární závislost a nezávislost)

Řekneme, že vektory $\vec{u}_1$, $\vec{u}_2$, ..., $\vec{u}_k$ jsou *lineárně závislé*, jestliže existuje alespoň jedna netriviální lineární kombinace těchto vektorů, jejímž výsledkem je nulový vektor $\vec{o}$, tj. existují-li reálná čísla t_1, t_2, ..., t_k, z nichž alespoň jedno je různé od nuly, taková, že platí

$$\vec{o} = t_1\vec{u}_1 + t_2\vec{u}_2 + \cdots + t_k\vec{u}_k.$$

V opačném případě říkáme, že vektory jsou *lineárně nezávislé*.

Bude-li z kontextu zřejmé, že proměnná je vektorem, budeme pro pohodlí šipku nad písmenem označujícím jméno proměnné vynechávat.

6.3 Matice

Definice (Matice)

Maticí řádu $m \times n$ rozumíme schema

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & \cdots & a_{mn} \end{pmatrix}$$

kde a_{ij} pro $i = 1..m$ a $j = 1..n$ jsou reálná čísla nebo funkce. Množinu všech matic řádu $m \times n$, jejichž prvky jsou reálná čísla, označujeme symbolem $\mathbb{R}^{m\times n}$. Zkráceně zapisujeme též $A = (a_{ij})$.

Je-li $m = n$ nazývá se matice A *čtvercová matice*, jinak *obdélníková matice*. Je-li A čtvercová matice, nazýváme prvky tvaru a_{ii}, tj. prvky, jejichž řádkový a sloupcový index jsou stejné, *prvky hlavní diagonály*.

Matice, která vznikne tak, že její řádky jsou tvořeny sloupci matice A se nazývá *matice transponovaná* k matici A a označuje A^T.

Pro matice definujeme *sčítání* a *násobení číslem* stejně jako u vektorů, tj. po složkách. Tyto operace přirozeně přebírají všechny důležité vlastnosti operace sčítání, jako jsou asociativita, komutativita, existence neutrálního prvku nebo existence opačného prvku.

V této fázi je vlastně jedno, jestli prvky jsou uspořádány jako řádkový nebo sloupcový vektor nebo jako matice. Odlišení matic a vektorů provedeme zavedením maticového součinu.

Definice (Součin matic)

Buďte $A = (a_{ij})$ matice řádu $m \times n$ a $B = (b_{ij})$ matice řádu $n \times p$. *Součinem matic* A a B (v tomto pořadí) rozumíme matici $G = (g_{ij})$ řádu $m \times p$, kde

$$g_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$$

pro všechna $i = 1..m$, $j = 1..p$. Zapisujeme

$$G = AB$$

(v tomto pořadí).

Slovy: v j-tém sloupci matice AB je lineární kombinace sloupců matice A, přičemž koeficienty této lineární kombinace jsou prvky z j-tého sloupce matice B.

Vztah pro maticový součin se také často zapisuje symbolicky

$$g_{ik} = \sum_{j=1}^{N} a_{ij}b_{jk}$$

nebo zkráceně

$$g_{ik} = a_{ij}b_{jk},$$

kde platí konvence, že pokud se nějaký index opakuje, potom se přes něj sčítá (Einsteinova sumační konvence). V tomto případě se vpravo opakuje index j a proto se pravé strana sčítá přes všechny hodnoty tohoto indexu.

Na maticový součin můžeme pohlížet i pomocí pojmů známých z analytické geometrie. Prvky v součinu matic jsou skalárními součiny řádků první matice se sloupci druhé matice.

Maticový součin má následující vlastnosti

- je asociativní

$$(AB)C = A(BC) = ABC,$$

- je distributivní vzhledem ke sčítání

$$A(B + C) = AB + AC \qquad \text{a} \qquad (B + C)A = BA + CA,$$

- není však komutativní (AB je obecně různé od BA, proto v předchozím máme roznásobování závorky zleva i zprava),

- ale při násobení skalárem komutativní je:

$$A(\lambda B) = \lambda(AB),$$

kde λ je reálné číslo a A a B jsou matice.

Můžeme tedy měnit uzávorkování, můžeme roznásobovat závorky, nesmíme však měnit pořadí matic při násobení.

6.4 Jednotková matice a matice inverzní

V tomto pododdíle si představíme maticovou obdobu jedničky jako neutrálního prvku vzhledem k násobení a převrácené hodnoty.

Definice (Jednotková matice)

Jednotková matice je čtvercová matice mající v hlavní diagonále jedničky a mimo hlavní diagonálu nuly. Značí se I.

Věta (Jednotková matice je neutrálním prvkem vzhledem k násobení)

Jednotková matice splňuje $AI = A$ a $IB = B$ pro libovolné matice A a B, pro které je součin definován.

Převrácená hodnota je definována tak, že součin čísla a jeho převrácené hodnoty je roven jedné. Maticová obdoba tohoto vztahu je v následující definici.

Definice (Inverzní matice)

Inverzní matice ke čtvercové matici A je matice A^{-1} splňující

$$AA^{-1} = A^{-1}A = I.$$

6.5 Matice jako zobrazení

Součin matice a vektoru je možno chápat jako zobrazení, které vektor zobrazuje na vektor. Vektorem bývá kvantitativní charakteristika populace v nějakém roce (rozebereme níže) a obrazem vektoru charakteristika v následujícím roce. Tímto způsobem je možné stav v jednom roce projektovat na stav v roce následujícím a proto se v tomto kontextu matice použitá v součinu nazývá matice projekce.

6.6 Vlastní čísla a směry

U zobrazování vektorů pomocí maticového násobení nás velice zajímá, které směry se zachovávají, tj. kdy bude obrazem vektoru jeho násobek.

Definice (Vlastní vektor a vlastní hodnota matice)

Řekneme, že nenulový vektor $\vec{u}$ je (pravým) *vlastním vektorem* matice A příslušným *vlastní hodnotě* λ, jestliže platí

$$A\vec{u} = \lambda\vec{u}.$$

Vlastní čísla se nazývají též vlastní hodnoty matice. Každý nenulový vlastní násobek vlastního vektoru je vlastní vektor příslušný téže vlastní hodnotě.

Kromě pravých vlastních vektorů někdy uvažujeme i levé vlastní vektory, definované rovností

$$\vec{v}\,A\ = \lambda\vec{v}$$

a v tomto případě je vektor $\vec{v}$ nutno chápat jako řádkový vektor.

Inverze matice, jejíž sloupce jsou tvořeny vlastními (pravými) vektory, má v řádcích vlastní levé vektory.

S vlastními směry se setkáme při hledání stabilních poměrů věkových skupin u věkově strukturované populace a při hledání řešení speciálních soustav lineárních rovnic, při hledání řešení autonomních systémů.

6.7 Markovův řetězec, model skladby lesa

Markovův řetězec je jeden z nejjednodušších modelů popisujících systém, který se může nacházet v různých stavech a mezi těmito stavy se náhodně přepíná podle předem daných pravděpodobností. Pro jeho popis je vhodný matematický aparát založený na teorii matic. Následující ukázka aplikace při studiu populací je z [3].

Americký vědec H. S. Horn studoval druhovou skladbu lesa a vycházel z předpokladů, že existuje konstantní pravděpodobnost, že určitý druh je nahrazen jiným druhem. Tabulka pravděpodobností je níže. Pro každý současný druh jsou v řádku pravděpodobnosti, že tento druh bude za 50 let nahrazen druhem ze záhlaví příslušného sloupce. Například pravděpodobnost toho, že na stanovišti, kde nyní roste bříza topololistá poroste za 50 let červený javor je 50% (první řádek, třetí sloupec). Pravděpodobnost toho, že na stanovišti, kde nyní roste javor za 50 let poroste bříza je nulová (třetí řádek, první sloupec). Model předpokládá, že i když se dřevina v lokalitě

nevyskytuje, existuje zdroj semen a dřevina se na této lokalitě může objevit.

	Bříza topololistá	Tupela lesní	Javor červený	Buk
Bříza topololistá	0.05	0.36	0.50	0.09
Tupela lesní	0.01	0.57	0.25	0.17
Javor červený	0.00	0.14	0.55	0.31
Buk	0.00	0.01	0.03	0.96

Procentuální zastoupení jednotlivých druhů budeme charakterizovat vektorem, kde hodnoty pro stromy budou ve stejném pořadí, jako jsou stromy seřazeny v naší tabulce. Pokud například je zastoupena napůl bříza a buk, odpovídá to vektoru $v(0) = (50, 0, 0, 50)^T$.

Procentuální zastoupení každého druhu se bude měnit z období na období. Například procentuální zastoupení javoru v dalším období bude dáno procentuálním zastoupením javoru v současnosti a pravděpodobností, že se na stanovišti udrží a dále procentuálním zastoupením ostatních dřevin a pravděpodobností, že tato dřevina bude nahrazena javorem. Tedy pro javor a vektor procentuálního zastoupení $v = (v_1, v_2, v_3, v_4)^T$ to bude

$$0.50v_1 + 0.25v_2 + 0.55v_3 + 0.03v_4$$

Při použití maticového součinu vektoru rozložení zastoupení s maticí pravděpodobností

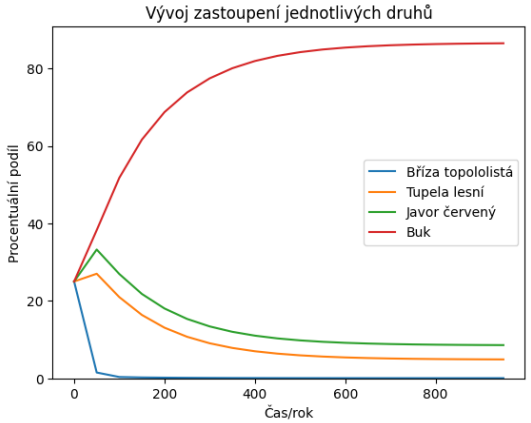
$$M = \begin{pmatrix} 0.05 & 0.01 & 0 & 0 \\ 0.36 & 0.57 & 0.14 & 0.01 \\ 0.50 & 0.25 & 0.55 & 0.03 \\ 0.09 & 0.17 & 0.31 & 0.96 \end{pmatrix}$$

je zastoupení dřevin v následujícím období dáno vzorcem

$$v(k+1) = Mv(k).$$

```
matrix([[0.05, 0.01, 0.  , 0.  ],
        [0.36, 0.57, 0.14, 0.01],
        [0.5 , 0.25, 0.55, 0.03],
        [0.09, 0.17, 0.31, 0.96]])
```

Zkusíme si namodelovat 20 období, tj. tisíc let vývoje. K tomu si připravíme pole do kterého budeme ukládat data. Výchozím stavem bude rovnoměrné zastoupení všech druhů. Vývoj jednotlivých dřevin zachytíme graficky.



Ze simulace se zdá, že poměr dřevin se postupně stabilizuje. Z matematického hlediska se procentuální zastoupení dřevin nemění, pokud je vektor zastoupení dřevin vlastním vektorem matice příslušným vlastní hodnotě 1. Ověříme, že to tak opravdu je.

```
[ 0.0516464   4.88304501  8.57619685␣
 ↪86.48911173]
[ 0.05141277  4.86748704  8.55816607␣
 ↪86.52293412]
```

Pokud bychom chtěli simulaci ne po 50 letech, ale po 100 letech, můžeme maticí vynásobit dvakrát. To je v konečném důsledku stejné, jako bychom násobili druhou mocninou. Pokud bychom chtěli delší časový interval, použijeme vyšší mocninu. Následující výpočet ukazuje, že pro dostatečně velkou mocninu vychází všechny sloupce matice stejné a jsou rovny výslednému poměru mezi jednotlivými dřevinami.

```
matrix([[0.00050824, 0.00050824, 0.
 ↪00050824, 0.00050824],
        [0.04828282, 0.04828282, 0.
 ↪04828281, 0.04828279],
        [0.0851273 , 0.0851273 , 0.
 ↪08512728, 0.08512726],
        [0.86608164, 0.86608164, 0.
 ↪86608167, 0.86608171]])
```

Další uplatnění Markovových řetězců je například při předpovědi počasí a jejím zpřesněním na lokální úroveň. Používá informace o tom, s jakou pravděpodobností je jeden druh počasí zachován či vystřídán druhým.

6.8 Leslieho model růstu populace s věkovou strukturou

Britský ekolog Patrick H. Leslie použil maticový součin v roce 1945 k formulaci modelu, sledujícího růst populace s definovanou věkovou strukturou. Model sleduje růst jednotlivých věkových skupin a přechod jedinců z jedné skupiny do druhé.

Model vysvětluje, proč se v populaci ustálí věkové složení na konstantních poměrech. Jedná se o model populární a úspěšný v ekologii a modelování populací, včetně populace lidské.

Model předpokládá, že populace je rozdělena do několika věkových kategorií a v každé kategorii je dána pravděpodobnost dožití se do další kategorie a průměrný počet potomků. Kromě toho je dán pro každou věkovou kategorii průměrný počet potomků připadajících na jednotlivce. Protože potomky rodí samice, byl tento model navržen a je zpravidla uvažován pro modelování počtu samic. Model je však možno adaptovat i na celou populaci, tj. nerozlišovat samce a samice.

Označíme-li pro populaci rozdělenou do tří věkových kategorií x_1 (nejmladší) až x_3 (nejstarší) průměrný počet potomků na jednotlivce (případně potomků samičího pohlaví) během časové jednotky hodnotami f_1 až f_3, je v dalším časovém okamžiku počet jedinců nejnižší kategorie dán výrazem

$$f_1x_1 + f_2x_2 + f_3x_3.$$

Je-li p_1 pravděpodobnost přežití členů nejmladší kategorie a jejich postupu do vyšší věkové třídy, je velikost populace v druhé věkové třídě dána výrazem

$$p_1x_1.$$

Podobně postupujeme při přechodu z prostřední do nejstarší kategorie. Zapsáno modelem i s explicitním uvedením času jako argumentu máme

$$\begin{aligned} x_1(k+1) &= f_1x_1(k) + f_2x_2(k) + f_3x_3(k) \\ x_2(k+1) &= p_1x_1(k) \\ x_3(k+1) &= p_2x_2(k) \end{aligned}$$

anebo (zapsáno jednou maticovou rovnicí)

$$\begin{pmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \end{pmatrix} = \begin{pmatrix} f_1 & f_2 & f_3 \\ p_1 & 0 & 0 \\ 0 & p_2 & 0 \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \end{pmatrix}.$$

Opakovaným násobením získáme věkovou strukturu populace v další generaci a toto se opakuje podobně jako u Markovova řetězce.

Při použití modelu volíme časový interval odpovídající jednotlivým stadiím dostatečně krátký tak, aby v prvním řádku byly alespoň dvě po sobě jdoucí hodnoty kladné. Potom má Leslieho matice následující vlastnosti.

- Matice má jednu kladnou vlastní hodnotu λ_1, která je dominantní v tom smyslu, že všechny ostatní vlastní hodnoty jsou v absolutní hodnotě menší.
- Dominantní vlastní hodnota určuje rychlost růstu populace v dlouhodobém měřítku.
  - Je-li dominantní vlastní hodnota rovna jedné, populace se v dlouhodobém měřítku stabilizuje a její velikost se ustálí na stavu, kdy dále neroste ani neklesá.
  - Je-li dominantní vlastní hodnota větší než jedna, velikost populace setrvale roste geometrickou řadou s exponentem daným touto vlastní hodnotou.
  - Je-li dominantní vlastní hodnota mezi nulou a jedničkou, velikost populace klesá geometrickou řadou.
- Vektor příslušný dominantní vlastní hodnotě určuje rozložení populace do jednotlivých věkových tříd poté, co se toto rozložení ustálí.

6.9 Zobecnění Leslieho modelu

6.9.1 Agregace nejstarších věkových kategorií

Pro praktické využití Leslieho modelu je někdy vhodné redukovat počet uvažovaných věkových tříd tak, že nejstarší věkové třídy shrneme do třídy jediné. Poté tedy členové nejstarší věkové třídy automaticky v dalším časovém kroku nevymírají, ale určité procento jich pouze zestárne a zůstávají ve své nejstarší kategorii. V Leslieho matici se toto projeví tak, že prvek v pravém dolním rohu matice je nenulový.

Model populace tuleňů

Následující příklad je z [12].

Populace samiček tuleňů je rozdělena na tři třídy: mláďata (0 až 4 roky), mladé tuleně (4 až 8 let) a dospělé tuleně (nad 8 let). Jednotka času budou 4 roky. Mláďata do čtyř let nemají potomky. Mladí tuleni mají průměrně 1.26 a dospělí průměrně 2.0 samičích potomků každé 4 roky. Mláďata dorostou do další věkové kategorie (tj. nezahynou) s pravděpodobností 0.614, mladí tuleni dorostou do kategorie dospělých s pravděpodobností 0.808 a pravděpodobnost, že starý tuleň přežije další 4 roky je také 0.808.

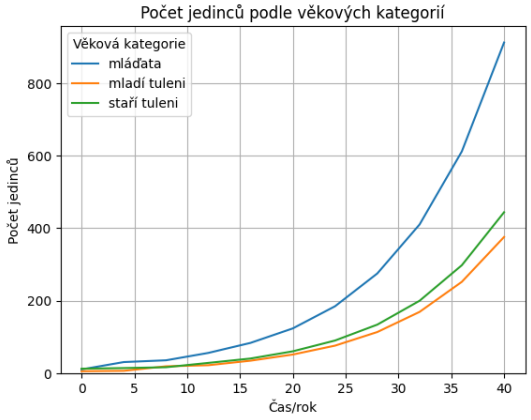
Model může být reprezentován rekurentním vztahem

$$\begin{pmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \end{pmatrix} = \begin{pmatrix} 0 & 1.26 & 2.0 \\ 0.614 & 0 & 0 \\ 0 & 0.808 & 0.808 \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \end{pmatrix}.$$

Matice $L = \begin{pmatrix} 0 & 1.26 & 2.0 \\ 0.614 & 0 & 0 \\ 0 & 0.808 & 0.808 \end{pmatrix}$ má jediné reálné kladné vlastní číslo a toto vlastní číslo je větší než 1.

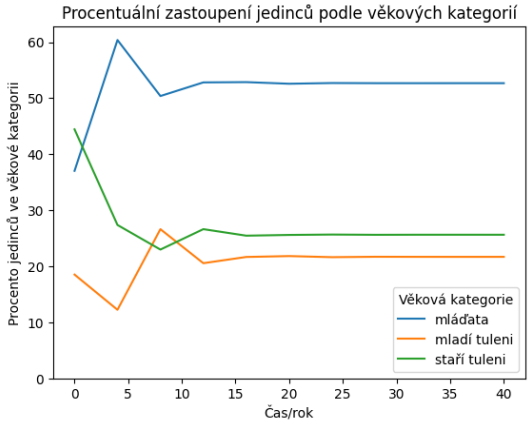
```
[[0.    1.26  2.   ]
 [0.614 0.    0.   ]
 [0.    0.808 0.808]]

Vlastní hodnoty:  [-0.34182332+0.
↪35954975j -0.34182332-0.35954975j
↪1.49164663+0.j        ]
Vlastní vektory jsou sloupce matice
 [[ 0.38392628-0.40383611j  0.
↪38392628+0.40383611j  0.84331389+0.
↪j         ]
 [-0.68962744+0.j          -0.
↪68962744-0.j          0.34712962+0.
↪j         ]
 [ 0.44144739+0.1380406j   0.
↪44144739-0.1380406j   0.4102715 +0.
↪j         ]]

Reálná vlastní hodnota je  1.
↪4916466348833581
Příslušný vlastní směr je [0.
↪52683575 0.2168591  0.25630515]
```

Časový vývoj modelu z počátečního stavu $[10, 5, 12]$ po dobu deseti jednotek času (tj. 40 let) je zachycen na následujícím obrázku.



Tip: Rychlost exponenciálního růstu se nejlépe posuzuje na grafu, kde svislá komponenta je logaritmická. Zkuste graf překreslit s novým nastavením, tj. vynechat omezení pro svislou osu a použít pro ni logaritmickou škálu. V takovém grafu má exponenciální funkce tvar přímky a ze vzájemné polohy přímek snadno poznáme, jestli všechny věkové kategorie rostou stejně rychle.

Pokud převedeme absolutní čísla na podíly z celku, získáme vývoj věkové skladby populace.



Procentuální skladba populace, ke které vývoj konverguje, je přibližně dána poslední položkou.

```
array([52.68328969, 21.68662839, 25.
↪63008192])
```

6.9.2 Lefkovitchův model

Anglický biolog L.P. Lefkovitch představil model analogický Leslieho modelu, ale namísto s věkovými kategoriemi model pracuje s vývojovými stadii. To je pro některé druhy přirozenější. Například rostliny v nepříznivých podmínkách mohou růst pomaleji a věk není spolehlivým prediktorem dalšího vývoje. Model je podobný Leslieho modelu s tím, že je jiná interpretace jednotlivých kategorií (nejedná se o věkové třídy, ale o vývojová stadia) a matice modelující chování takového systému může mít na rozdíl od Leslieho matice nenulové prvky i v hlavní diagonále.

U živočichů se tento model může používat v situacích, kdy jednotlivé fáze života nejsou stejně dlouhé, například u želv (Crouse, Crowder, Caswell: A stage-based population model for loggerhead sea turtles and implications for conservation, Ecology, 68(5), 1987, pp. 1412-1423, http://web.abo.fi/fak/mnf/mate/kurser/matriser/Turtlearticle.pdf). Dále se používá v případě, že je obtížné nebo neefektivní stanovit věk jednotlivců nebo v případě, kdy někteří jedinci vykazují zrychlený či zpomalený růst.

Zobecnění Leslieho matice uvedené v předchozí podkapitole je vlastně speciálním případem Lefkovitchova modelu.

6.9.3 Usherův model

S modifikací podobnou Lefkovitchově modelu přišel britský biolog Michael B. Usher. Namísto vývojových stadií použil jako prediktor dalšího vývoje velikost. To je vhodné pro některé rostliny, například pro stromy.

Společným znakem všech dosud uvedených modelů (Markov, Leslie, Lefkovitch, Usher) je, že matice je použita jako nástroj projektující současný stav populace na situaci po uplynutí jedné časové jednotky.

6.10 Soustavy lineárních rovnic

Uvažujme soustavu lineárních rovnic

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &= b_1, \\ a_{21}x_1 + a_{22}x_2 &= b_2. \end{aligned} \tag{6.1}$$

Tuto soustavu je možné zapsat jako jednu vektorovou rovnici ve tvaru

$$\begin{pmatrix} a_{11} \\ a_{21} \end{pmatrix} x_1 + \begin{pmatrix} a_{21} \\ a_{22} \end{pmatrix} x_2 = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

anebo pomocí maticového součinu ve tvaru

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Označíme-li matice v tomto vztahu postupně A, X a B, potom soustavu (6.1) je možno zapsat ve tvaru

$$AX = B.$$

Tuto možnost je pochopitelně možno použít pro libovolný počet rovnic a neznámých. Maticový součin umožňuje snadno zapsat a pracovat se soustavami lineárních rovnic.

6.11 Lineární autonomní systémy

Lineární autonomní systémy jsou soustavy ve tvaru

$$\frac{\mathrm{d}X}{\mathrm{d}t} = AX + B,$$

kde X je neznámá vektorová funkce, B je vektor a A je čtvercová matice.

Je-li dimenze vektorů malá, bývá zvykem systém rozepsat do komponent. Například ve dvoudimenzionálním případě by tento zápis byl ekvivalentní dvojici rovnic

$$\begin{aligned} \frac{\mathrm{d}x_1}{\mathrm{d}t} &= a_{11}x_1 + a_{12}x_2 + b_1, \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} &= a_{21}x_1 + a_{22}x_2 + b_2. \end{aligned}$$

S takovými soustavami se setkáme při studiu interagujících populací. Výhodou lineárních autonomními systémů je, že pomocí vlastních čísel a vlastních vektorů můžeme napsat přímo jejich řešení. Nevýhodou je, že reálné děje jsou komplikovanější, než je podchytitelné lineárními rovnicemi a jejich soustavami. Neznamená to však, že lineární rovnice a soustavy rovnic nevyužijeme. Využívá se jich naopak velmi často. Jsou to totiž jediné soustavy diferenciálních rovnic, které umíme vyřešit. Proto se na ně spoléháme i při práci s nelineárními modely. Princip práce v takovém případě spočívá v tom, že vytipujeme body našeho zájmu v nich aproximujeme nelineární model modelem lineárním. Poté vyřešíme lineární model a nakonec informace o řešení přeneseme na model nelineární.

KAPITOLA 7

Parciální derivace

Co se dozvíte v tomto textu

V této části se naučíme sledovat rychlosti růstu funkce dvou a více proměnných.

Mezi aplikace patří ochrana ohrožených druhů, kdy při snaze podpořit růst populace musíme vybrat ze škály parametrů ovlivňujících vývoj ten, který má na růst populace nejpodstatnější vliv. Postup je možné použít i naopak a při snaze o kontrolu škůdců vytipovat parametr, který vývoj populace škůdce nejvíce postihne.

Další aplikací je model šíření invazních druhů, kdy se zastoupení živočišného druhu mění v čase i v prostoru tím, jak se druh množí a rozšiřuje území, na kterém žije.

Foto: Ondatra se jako invazní druh velmi rychle rozšířila z Dobříšského panství do celé Evropy. Tato invaze byla popsána v jednom z prvním článků věnovaných invazím živočišných druhů a díky tomu se naše země dostala do většiny ekologických učebnic. D. Gordon E. Robertson, https://commons.wikimedia.org/

Pomocí derivace jsme se naučili sledovat rychlost růstu funkce jedné proměnné. Tato veličina je užitečná protože rychlost růstu je často determinována měřitelnými veličinami a je k dispozici zákon vyjadřující souvislost rychlosti růstu sledované veličiny s ostatními parametry systému. Často je mezi těmito parametry i ona sledovaná veličina, což při sestavení modelu vede na diferenciální rovnice.

Někdy není možné si vystačit pouze s funkcemi jedné proměnné. Proto si nyní koncept derivace rozšíříme i na funkce více proměnných.

7.1 Zavedení parciální derivace

Změna funkce více proměnných může být způsobena změnou libovolné nezávislé proměnné. Pokud sledujeme například rychlost růstu populace popsané Leslieho maticí, může nás zajímat, jak tato rychlost růstu souvisí s jednotlivými prvky projekční matice, tj. s plodností jednotlivých věkových tříd a s jejich pravděpodobností přežití. Zdá se býti rozumné oddělit jednotlivé změny a studovat každou samostatně. Buď v daném bodě fixovat všechny parametry až na jeden a sledovat například vliv pravděpodobnosti přežití u nejmladší věkové kategorie na celkový růst populace.

Následující definice výše uvedenou myšlenku odděleného sledování změny funkce (závislé veličiny) jako reakce na změnu jedné jediné vstupní informace (jedné nezávislé veličiny) uvádí v život. Definice je stejná jako u derivace funkce jedné proměnné, změna je pouze v tom, že je přítomna i další proměnná.

Definice (Parciální derivace funkce dvou proměnných)

Buď $f\colon \mathbb{R}^2 \to \mathbb{R}$ funkce dvou proměnných, x a y, tj. $f(x,y)$. Výraz

$$\frac{\partial f}{\partial x} := \lim_{h\to 0} \frac{f(x+h,y)-f(x,y)}{h}$$

se nazývá *parciální derivace funkce f podle x*. Podobně,

$$\frac{\partial f}{\partial y} := \lim_{h\to 0} \frac{f(x,y+h)-f(x,y)}{h}$$

je *parciální derivace funkce f podle y*.

Podobně můžeme definovat parciální derivaci pro funkce libovolného konečného počtu proměnných. V těchto parciálních derivacích vlastně sledujeme, jak reaguje veličina f na změny jenom v jedné proměnné. Proměnná, přes kterou se nederivuje, má vlastně roli parametru, nijak se nemění.

Kromě $\frac{\partial f}{\partial x}$ se běžně parciální derivace označuje symboly f'_x, f_x, $\partial_x f$ nebo $D_x f$.

Parciální derivaci $\frac{\partial f}{\partial x}$ je tedy možné chápat jako rychlost s jakou se mění veličina f jako funkce proměnné x. Ekvivalentně jako změnu veličiny f při změně veličiny x o jednotku. Tato hodnota bude malá, pokud se funkční hodnoty při změnách x moc nemění. To by znamenalo, že rozkolísání hodnot veličiny x má malý vliv na veličinu f. V tomto smyslu se jedná o jakousi míru citlivosti (sensitivity) a parciální derivace se používají pro citlivostní analýzu.

Vektor

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial x}\right)$$

se nazývá gradient funkce f. Gradient míří směrem, ve kterém roste funkce f nejrychleji a jeho délka je rovna nárůstu funkční hodnoty na jednotce délky. V přírodních vědách často pracujeme se záporně vzatým gradientem, který udává směr a intenzitu maximálního poklesu funkčních hodnot funkce f.

7.2 Lineární aproximace

Jedna z úspěšných inženýrských metod je nahrazení nelineární funkce pomocí funkce lineární. Není v tom nic složitějšího než prostá selská logika představená v následující poznámce.

Poznámka (Lineární aproximace selskou logikou)

Uvažujme modelový případ populace jelenů, která má v roce 2020 celkem 1023 členů a roste rychlostí 12 jedinců za rok. (Čistý růst, tedy přírůstek snížený o úmrtí a lov.) Odhadneme velikost populace v budoucích letech. V roce x je populace celkem $\Delta x = x - 2020$ let od roku 2020. V tomto roce bude velikost $N(x)$ populace dána součtem velikosti v roce 2020 a přírůstku za daný počet let. Tento přírůstek je roven přírůstku za rok vynásobenému počtem let, tedy

$$\Delta N = 12\Delta x.$$

Velikost populace v roce x je

$$N(x) = N(2020) + \Delta N = 1023 + 12\Delta x = 1023 + 12(x-2020).$$

V předchozím příkladě je rychlost růstu vlastně derivace velikosti populace podle času. Stejným postupem můžeme určit časový vývoj libovolné veličiny, u níž je znám výchozí stav a rychlost růstu. Má to pouze jednu vadu na kráse. Málokdy se totiž setkáme s tím, že rychlost růstu je konstantní. Proto uvedená myšlenka bude platit jenom přibližně a jenom lokálně v okolí bodu, z nějž vycházíme. To proto, že na krátkém intervalu se rychlost růstu funkce zpravidla nestihne podstatně změnit. Dostáváme následující přibližné vzorce pro změnu funkce a pro funkční předpis v okolí bodu aproximace.

$$\Delta f \approx \frac{\mathrm{d}f}{\mathrm{d}x}(x_0)\Delta x$$

$$f(x) \approx f(x_0) + \frac{\mathrm{d}f}{\mathrm{d}x}(x_0)(x-x_0)$$

Geometricky uvedená aproximace znamená, že křivku definovanou grafem funkce nahrazujeme přímkou. Tímto jsou dány i meze použitelnosti: metoda výborně funguje pro funkce, jejichž derivace se mění pomalu a graf připomíná lineární závislost. Kvantitativně to je možno vyjádřit tak, že derivace má malou derivaci, tj. že druhá derivace je malá (ve smyslu numericky blízko k nule, tj. že absolutní hodnota druhé derivace je malá).

Pokud funkce závisí na dvou nebo více proměnných, jednotlivé příspěvky ke změně sečteme.

$$\Delta f \approx \frac{\partial f}{\partial x}(x_0,y_0)\Delta x + \frac{\partial f}{\partial x}(x_0,y_0)\Delta y$$

$$f(x,y) \approx f(x_0,y_0) + \frac{\partial f}{\partial x}(x_0,y_0)(x-x_0) + \frac{\partial f}{\partial x}(x_0,y_0)(y-y_0)$$

Toto je možno zapsat vektorově pomocí gradientu a součinu řádkové a sloupcové matice ve tvaru

$$f(x,y) \approx f(x_0,y_0) + \nabla f(x_0,y_0)\begin{pmatrix} x-x_0 \\ y-y_0 \end{pmatrix}.$$

Je-li $\vec{F}(x,y) = (f_1(x,y), f_2(x,y))$ vektorová funkce dvou proměnných, je možné napsat linearizaci podle výše uvedeného vzorce pro každou komponentu samostatně a v maticovém zápisu máme

$$\vec{F}(x,y) = \vec{F}(x_0,y_0) + J(x_0,y_0)\begin{pmatrix} x-x_0 \\ y-y_0 \end{pmatrix},$$

kde

$$J(x,y) = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{pmatrix}$$

je Jacobiho matice zobrazení $\vec{F}$

Jacobiho matice má tedy v řádcích gradienty jednotlivých komponent.

7.3 Sensitivita a elasticita projekční matice

Zkoumejme model

$$x(n+1) = Ax(n)$$

věkově nebo jinak strukturované populace, kde $A = (a_{ij})$ je projekční matice zobrazující současný stav na stav v následujícím časovém okamžiku. Bude nás zajímat, jak se bude měnit dynamika modelu při změnách parametrů. To je důležité pro nastavení správných postupů při ochraně živočišných druhů nebo při nastavení dlouhodobě udržitelné intenzity lovu či sklizně.

Rychlost růstu populace je dána dominantní vlastní hodnotou matice A. Derivace $\frac{\partial\lambda}{\partial a_{ij}}$ dominantní vlastní hodnoty λ podle prvku a_{ij} udává, jak rychle se mění tato vlastní hodnota při změnách prvku a_{ij}. Jedná se o *citlivost* (*sensitivitu*) dominatní vlastní hodnoty na komponentu a_{ij} projekční matice A. Podle [4] platí

$$\frac{\partial\lambda}{\partial a_{ij}} = v_i w_j,$$

kde w a v jsou pravý a levý vlastní vektory příslušné vlastní hodnotě λ a jejichž skalární součin je roven jedné.

Změna $\Delta\lambda$ vlastní hodnoty způsobená změnou parametru a_{ij} o Δa_{ij} je

$$\Delta\lambda = \frac{\partial\lambda}{\partial a_{ij}}\Delta a_{ij}.$$

Matice $\left(\frac{\partial\lambda}{\partial a_{ij}}\right)$ se nazývá *matice sensitivity* nebo *matice citlivosti dominantní vlastní hodnoty na složky projekční matice*. Matice ukazuje, jaké mají jednotlivé parametry populace vliv na celkový růst. Zpravidla ji počítáme pomocí sloupcového pravého vlastního vektoru $\vec{w}$ a řádkového levého vlastního vektoru $\vec{v}$ pomocí vztahu

$$\left(\frac{\partial\lambda}{\partial a_{ij}}\right) = \vec{v}^T\vec{w}^T,$$

přičemž tyto vektory splňují $\vec{v}\,\vec{w} = 1$ (skalární součin je roven jedné). Podmínka na skalární součin bývá splněna automaticky, pokud pokud levý vlastní vektor najdeme pomocí inverzní matice složené z pravých vlastních vektorů.

Často je vhodné pracovat s relativními změnami $\frac{\Delta\lambda}{\lambda}$ a $\frac{\Delta a_{ij}}{a_{ij}}$, pro které plyne vztah

$$\frac{\Delta\lambda}{\lambda} = \frac{\partial\lambda}{\partial a_{ij}}\frac{a_{ij}}{\lambda}\frac{\Delta a_{ij}}{a_{ij}},$$

kde výraz

$$\frac{\partial\lambda}{\partial a_{ij}}\frac{a_{ij}}{\lambda}$$

se nazývá *elasticita* dominantní vlastní hodnoty vzhledem ke složce a_{ij}. Udává, o kolik procent se změní maximální vlastní číslo, pokud se koeficient a_{ij} změní o jedno procento. Součet všech elasticit je roven jedné a proto je možno elasticitu chápat jako míru, s jakou se jednotlivé komponenty matice projekce podílejí na růstu populace. Bývá výhodné elasticity uspořádat do matice

$$\left(\frac{\partial\lambda}{\partial a_{ij}}\frac{a_{ij}}{\lambda}\right),$$

která se nazývá *matice elasticity*.

Prozkoumejme matici elasticity na příkladu s populací tuleňů z minulé přednášky.

```
matrix([[0.        , 0.10157076, 0.
↪19054952],
        [0.29212028, 0.        , 0. ␣
↪       ],
        [0.        , 0.19054952, 0.
↪22520993]])
```

Vidíme, že největší vliv na růst populace mají parametry spojené s přežíváním nejstarší věkové kategorie a s dospíváním mláďat, toto jsou dvě jediné hodnoty nad 20 procent. Pokud populace vymírá, snažíme se při ochraně tohoto druhu navýšit parametry s nejvyšší elasticitou. Změna těchto parametrů má totiž největší efekt. Fertilita jednotlivých věkových tříd (čísla v prvním řádku) nemá tak výrazný vliv a navíc ji zpravidla nemůžeme tak efektivně ovlivnit, jako koeficienty přežití.

Poznámka (Matice elasticity v ochraně přírody)

Podle [15] (str. 127) by se v ochraně přírody při ochraně ohrožených druhů měly navyšovat ty koeficienty, které jsou spojeny s největšími elasticitami. Při redukci populace škůdců se také věnujeme koeficientů s největšími elasticitami, ale tyto koeficienty se snažíme snižovat. Pokud populaci hospodářsky využíváme, poté lov či sklizeň zaměříme naopak na kategorie, kde jsou koeficienty s nejnižšími elasticitami. V těchto případech totiž vnější zásahy ovlivňují populaci relativně nejméně. Vždy je však nutné přihlédnout k tomu, že některé koeficienty ovlivníme relativně snadno, některé jenom s velkými ekonomickými náklady a některé prakticky ovlivnit moc nemůžeme.

Pro kontrolu můžeme vypočítat matici elasticity po složkách pomocí numerické aproximace derivace centrální diferencí. S každou komponentou matice posuneme o zvolený krok nahoru a dolů, v každém z těchto případů vypočteme dominantní vlastní hodnotu a určíme derivaci (rychlost změny) pomocí poměru změn a poté vypočteme elasticitu jako relativní rychlost změny. Přitom není nutné počítat derivace pro nulové komponenty matice, protože tyto derivace budou při přepočtu na relativní rychlost změny násobeny nulou a neuplatní se.

```
matrix([[0.        , 0.10157068, 0.
↪19055025],
        [0.29212876, 0.        , 0.
↪        ],
        [0.        , 0.19055404, 0.
↪22521132]])
```

Součet komponent matice elasticity by měl být v rámci zaokrouhlovací chyby roven jedné.

```
1.0
```

Pokud potřebujeme regulovat rychlost růstu populace, má smysl soustředit se na ty komponenty matice elasticity, které jsou numericky největší. Jejich změna se na růstu populace projeví nejvíce. (Některé koeficienty projekční matice je ovšem z principu těžké nebo nemožné regulovat, potom se soustředíme na ty, které regulovat můžeme.)

Koeficienty matice elasticity můžeme navzájem sčítat a součty interpretovat jako agregovaný podíl na růstu populace. Například součty po sloupcích odhalí, jak se jednotlivé věkové kategorie podílejí na celkovém růstu.

```
matrix([[0.29212876, 0.29212473, 0.
↪41576157]])
```

7.4 Rovnice vedení tepla

Rovnice vedení tepla popisuje vedení tepla v jednorozměrném objektu, například v tyči. Teplo se předává z teplejšího místa do místa o nižší teplotě a tím se v čase teplejší místo ochlazuje a chladnější otepluje.

Vedení tepla je proces, který si většina z nás umí snadno představit a má s ním praktické zkušenosti. Model je však plně univerzální a hodí se pro libovolný transportní děj v přírodě. To zahrnuje nejenom transport energie ve formě tepla, ale i transport látek, například pomocí difuze nebo prouděním v pórovitém prostředí. Tím je podchyceno například šíření nečistot ze zdroje do okolí, proudění podzemní vody, sušení dřeva ale i rozšiřování chorob či invazních druhů v ekosystému. To si ukážeme později (Fisherova–KPP rovnice).

Shrňme si nejprve základní fyzikální představy o vedení tepla.

- Rychlost růstu teploty v čase je úměrná rychlosti, s jakou je do daného místa dodáváno teplo. Pokud do daného místa dodáváme teplo, teplota roste tím rychleji, čím více tepla dodáváme za jednotku času.
- Rychlost, s jakou je do daného místa dodáváno teplo, se určí jako pokles toku tepla podél tyče v daném místě.
- Tok tepla je úměrný rychlosti, s jakou klesá teplo podél tyče.

Rychlost růstu je vždy vyjádřena derivací, rychlost poklesu záporně vzatou derivací. Pokud sledujeme, jak se veličina mění v čase, jedná se o derivaci podle času, pokud sledujeme změnu podél tyče, jedná se o derivaci podle polohy. Tímto jsme si připravili vše potřebné k formulaci matematického modelu.

Teplota T souvisí s polohou a s časem, jedná se tedy o funkci dvou proměnných a pro vyjádření rychlosti změny musíme použít parciální derivace. Parciální derivace teploty podle času $\frac{\partial T}{\partial t}$ udává, jak rychle s daném místě teplota roste s časem a souvisí s rychlostí dodávání tepla. Rychlost dodávání tepla je možné měřit tím, jak v daném místě slábne tok tepla q, tj. jaká je záporně vzatá derivace toku tepla podle polohy. Konstanty úměrnosti dodá fyzika, jedná se o hustotu ϱ a měrnou tepelnou kapacitu c, tj.

$$\varrho c \frac{\partial T}{\partial t} = -\frac{\partial q}{\partial x}.$$

Teplo teče z místa s větší teplotou do místa s menší teplotou a teče intenzivněji, pokud je mezi těmito místy velký teplotní rozdíl. Pokles tepla podél tyče je dán záporně vzatou derivací teploty podle polohy a tok je tomuto poklesu úměrný (s konstantou úměrnosti, která se nazývá specifická vodivost a značí λ). Tedy

$$q = -\lambda \frac{\partial T}{\partial x}.$$

Spojením obou vztahů dostáváme rovnici vedení tepla ve tvaru

$$\varrho c \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(\lambda \frac{\partial T}{\partial x}\right).$$

Je-li λ konstantní (nezávislá na teplotě a poloze), potom je možné rovnici upravit na

$$\varrho c \frac{\partial T}{\partial t} = \lambda \frac{\partial^2 T}{\partial x^2},$$

kde $\frac{\partial^2 T}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial T}{\partial x}\right)$ je druhá derivace teploty podle polohy.

7.5 Fisherova–KPP rovnice

Analogický postup jako u vedení tepla funguje pro libovolný transportní děj. Používá se například pro model šíření živočišného druhu v životním prostředí nebo šíření genu v populaci. Ukážeme si zde lineární verzi pro jednorozměrné životní prostředí, například podél řeky. Vícerozměrné zobecnění si uvedeme na konci semestru.

Tento model se nazývá Fisherova–KPP rovnice a jedná se vlastně o rovnici vedení tepla, do které jsou doplněny zdroje.

Fisherova–KPP rovnice popisuje populaci, která se může šířit v prostoru. Kromě časové závislosti tedy musíme uvažovat i závislost na poloze. Tato rovnice je vyjádřena pro funkci vyjadřující hustotu populace. Hustotou populace může být například množství jedinců na jednotku délky, v případě zobecnění na více dimenzí množství jedinců na jednotku plochy. Procesy vedoucí k migraci a tedy i změně hustoty populace jsou stejné jako u vedení tepla. Populace má tendenci migrovat z míst, kde je vysoká hustota do míst, kde je hustota menší a v místě, kde převažuje imigrace nad emigrací velikost populace roste. Protože se populace může rozmnožovat, je do rovnice navíc započítán člen modelující vlastní reprodukci. Zpravidla je uvažován logistický růst a rovnice má poté tvar

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + ru\left(1 - \frac{u}{K}\right).$$

Tato rovnice byla původně navržena jako model šíření výhodného genu populací. To potvrzuje, že v rovnici máme opravdu nástroj pro šíření nebo transport různých objektů, od energie, přes molekuly nebo částice v látce či geny v populaci až po jedince invazního druhu v ekosystému.

KAPITOLA 8

Autonomní systémy

Co se dozvíte v tomto textu

V této kapitole zkombinujeme aparát lineární algebry a diferenciálních rovnic a budeme se věnovat problematice soustav diferenciálních rovnic. Jedna z aplikací této teorie je modelování interakce dravce a kořisti. Pochopení principů této interakce je zásadní nejenom pro ochranu populace dravce či jeho kořisti, ale také pro zajištění toho, že případné zásahy do této rovnováhy nebudou mít nečekaný efekt. Seznámíme se s takzvaným Volterrovým efektem, kdy se snaha vymýtit nepohodlný živočišný druh může obrátit v populační explozi tohoto druhu.

Foto: Medvěd ulovil kořist. Model dravce a kořisti (nebo analogické modely býložravce a rostliny, případně parazita a hostitele) je jedním ze základních ekologických modelů. Autor Jean Beaufort, https://www.publicdomainpictures.net/.

Autonomní systémy diferenciálních rovnic jsou soustavy diferenciálních rovnic, kde každá rovnice obsahuje na levé straně derivaci jedné závislé proměnné a na pravé straně funkci všech závislých proměnných. Nezávislou proměnnou je zpravidla čas. To znamená, že pravá strana nezávisí na čase. V případě dvourozměrného autonomního systému se tedy jedná o soustavu diferenciálních rovnic ve tvaru

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= f(x,y), \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= g(x,y), \end{aligned} \tag{8.1}$$

kde f a g jsou funkce dvou proměnných. Řešením je dvojice funkcí $x(t)$ a $y(t)$, které vyhovují těmto rovnicím. Počáteční podmínku zpravidla formulujeme ve tvaru $x(0) = x_0$ a $y(0) = y_0$, kde x_0 a y_0 jsou reálná čísla.

8.1 Lineární autonomní systémy

Soustava n diferenciálních rovnic tvaru

$$\frac{\mathrm{d}X}{\mathrm{d}t} = AX + B,$$

kde X a B jsou n-rozměrné sloupcové vektory a A je n-rozměrná čtvercová matice je lineární autonomní systém.

Je-li X_0 řešením rovnice

$$AX + B = 0,$$

je konstantní funkce $X(t) = X_0$ řešením této rovnice. Toto řešení se nazývá *stacionární bod*.

Protože po substituci $Y = X - X_0$ se rovnice transformuje na rovnici

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = AY,$$

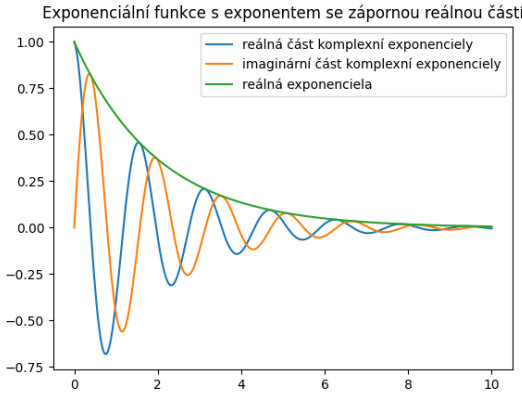
můžeme bez újmy na obecnosti předpokládat, že B je nulový vektor a studovat rovnici ve tvaru

$$\frac{\mathrm{d}X}{\mathrm{d}t} = AX \tag{8.2}$$

se stacionárním bodem v počátku. Rovnice má následující vlastnosti.

- Lineární kombinace libovolného počtu řešení rovnice (8.2) je také řešením této rovnice.
- Je-li λ vlastním číslem matice A a U je příslušný vlastní vektor, je funkce $X(t) = e^{\lambda t}U$ řešením rovnice (8.2). Jsou-li λ a U komplexní, je řešením rovnice samostatně reálná i imaginární část.

Chování řešení pro případ záporného vlastního čísla nebo čísla se zápornou reálnou částí charakterizuje následující graf. Zjednodušeně řečeno, řešení konverguje k nule buď monotonně (reálná vlastní hodnota) nebo v oscilacích (komplexní vlastní hodnota).



Experimentem s uvedeným kódem je možné ověřit, že vetší imaginární část se projeví zrychlením oscilací a více záporná reálná část se projeví zrychlením poklesu. Podobné grafy je možno nakreslit i pro kladnou vlastní hodnotu nebo pro vlastní hodnotu s kladnou reálnou částí. Rozdíl je v tom, že v takovém případě řešení buď roste do nekonečna, nebo osciluje se stále větší amplitudou.

8.2 Nelineární autonomní systémy

Soustava

$$X' = F(X)$$

diferenciálních rovnic, kde pravé strany nezávisí na čase, se nazývá autonomní systém.

Je-li $F(X_0) = 0$, je možno systém v okolí bodu X_0 aproximovat lineárním systémem

$$X' = J(X_0)(X - X_0),$$

kde $J(X_0)$ je Jacobiho matice funkce $F(X)$ v bodě X_0, tj. pro $F(X) = (F_1(X), \dots, F_n(X))^T$ je

$$J(X_0) = \left(\frac{\partial F_i(X_0)}{\partial x_j} \right).$$

O chování trajektorií v okolí stacionárního bodu tedy rozhodnou vlastní čísla Jacobiho matice. Za předpokladu, že jsme relativně daleko od případů, kdy se mění typ stacionárního bodu, tj. vlastní čísla jsou navzájem různá, jsou nenulová a v případě komplexních vlastních čísel mají nenulové reálné části, má původní nelineární systém stejný typ stacionárního bodu jako lineární systém s Jacobiho maticí. Nelineární systém tedy v jistém smyslu „zdědí“ chování řešení od své lineární aproximace pomocí Jacobiho matice. Je však nutno připomenout, že aproximace pomocí Jacobiho matice je jenom lokální a můžeme takto posoudit jenom řešení z nějakého okolí stacionárního bodu.

Zejména tedy, pokud má Jacobiho matice všechny vlastní hodnoty záporné, tak všechna řešení z nějakého okolí stacionárního bodu konvergují do tohoto bodu. Pokud má všechny vlastní hodnoty kladné, všechna řešení z nějakého okolí se naopak od stacionárního bodu vzdalují. To platí i pro vlastní komplexní vlastní hodnoty, pouze se mezi konvergencí a vzdalování přepíná podle znaménka reálné části vlastních hodnot a řešení oscilují směrem ke stacionárnímu bodu nebo od něj.

8.3 Autonomní systémy v rovině

Pro autonomní systém v rovině (8.1) je možné chápat řešení jako parametricky zadané rovinné křivky. Tyto křivky se nazývají trajektorie. Trajektorie buď konvergují do nějakého stacionárního bodu, nebo mají alespoň jednu ze složek neohraničenou, nebo konvergují k nějakému cyklu. Vektorové pole definované vektorovou funkcí na pravé straně autonomního systému je tvořeno šipkami, které míří stejným směrem jako trajektorie. Toto pole se nazývá směrové pole autonomního systému a umožňuje odhadnout chování trajektorií a tím i chování řešení. Na rozdíl od směrového pole diferenciální rovnice, směrové pole autonomního systému může obsahovat i šipky mířící doleva.

Vizualizaci řešení autonomního systému tedy můžeme provést dvěma způsoby.

- Vykreslit obě komponenty řešení jako funkce nezávislé proměnné. Výsledkem jsou tedy dvě křivky udávající časový průběh každé z komponent řešení.
- Vykreslit řešení jako parametrickou křivku. Výsledkem je tedy jedna křivka, ze které jsou patrné stavy, kterými systém postupně prochází, ale nemáme informaci o čase. Tato varianta je běžně používána a proto ji rozebereme níže.

Obecný tvar autonomního systému v rovině je

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= f(x, y) \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= g(x, y) \end{aligned}.$$

Křivka nebo křivky vyhovující rovnici

$$f(x, y) = 0$$

se nazývají x-nulkliny a v průsečíku trajektorie a x-nulkliny míří trajektorie svisle (hodnota x se mění nulovou rychlostí).

Křivka nebo křivky vyhovující rovnici

$$g(x, y) = 0$$

se nazývají y-nulkliny a v průsečíku trajektorie a y-nulkliny míří trajektorie vodorovně (hodnota y se mění nulovou rychlostí).

V průsečíku nulklin je stacionární bod.

Jacobiho matice

$$J(x, y) = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{pmatrix}$$

definuje lineární systém, který má v okolí každého bodu stejné chování trajektorií jako systém nelineární. Toto má smysl dělat ve stacionárních bodech a podle vlastních čísel Jacobiho matice v těchto stacionárních bodech rozhodneme o chování trajektorií. (Mimo stacionární body je situace triviální, trajektorie míří směrem definovaným pravou stranou systému a proto linearizaci nepotřebujeme.)

Vlastní čísla Jacobiho matice ve stacionárním bodě jsou dvě a pokud jsou komplexními čísly, potom mají stejnou reálnou část a imaginární část se liší znaménkem. To znamená, že je jenom několik málo druhů možného chování trajektorií v okolí stacionárních bodů. To dovoluje stacionární body klasifikovat podle toho, zda přitahují či odpuzují trajektorie ze svého okolí a zda toto přitahování či odpuzování souvisí s přikloněním se k nějakému pevnému směru, nebo zda je realizováno oscilacemi s měnící se amplitudou.

- Stabilní uzel je stacionární bod takový, že pro $t \to \infty$ všechny trajektorie z nějakého okolí konvergují do tohoto bodu bez oscilací. Nestabilní uzel má stejnou vlastnost, ale pro $t \to -\infty$, tedy trajektorie z tohoto bodu vycházejí. Stabilní uzel poznáme podle dvou záporných a nestabilní uzel podle dvou kladných reálných vlastních hodnot.
- Stabilní a nestabilní ohnisko je stacionární bod se stejnou vlastností jako uzel, ale konvergence je spojena s oscilacemi okolo stacionárního bodu. Stabilní ohnisko poznáme podle dvou komplexně sdružených vlastních hodnot se zápornou reálnou částí, nestabilní ohnisko s kladnou reálnou částí.
- Sedlo je stacionární bod, který má v každém okolí pouze konečný počet trajektorií, které pro $t \to \pm\infty$ konvergují k tomuto bodu. Poznáme jej podle jedné kladné a jedné záporné vlastní hodnoty.
- Bod rotace je takový bod, v jehož každém okolí jsou cykly. Pokud navíc v nějakém okolí existují pouze cykly, nazývá se tento bod navíc střed. Bod rotace souvisí s komplexně sdruženými vlastními čísly s nulovou reálnou částí, ale v těchto případech může stacionární bod být i ohniskem.

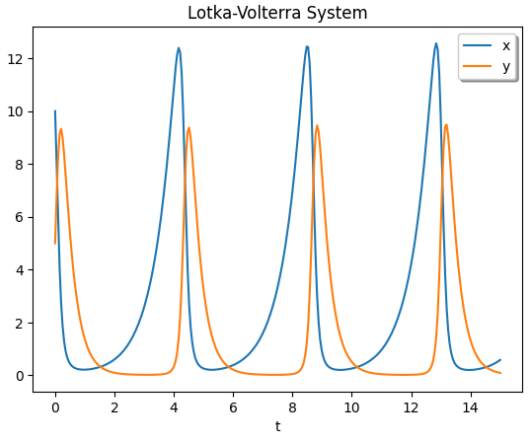
8.4 Lotkův–Volterrův model dravce a kořisti

Jednoduchým modelem dravce a kořisti je Lotkův–Volterrův model, který byl sestaven k vysvětlení fenoménu, že při omezení rybolovu během první světové války se v úlovcích zvýšilo procento dravých ryb. Tedy, že omezení rybolovu vedlo k nárůstu populace predátorů. Jinými slovy, lov má pozitivní vliv na populaci kořisti (její relativní velikost vzhledem k populaci dravce), zatímco ukončení lovu má pozitivní vliv na populaci dravce (opět ve smyslu relativní velikosti v porovnání s populací kořisti). Tento efekt se nazývá Volterrův efekt a je jednoduchým důsledkem vzájemné interakce mezi oběma živočišnými druhy.

Model vychází z jednoduché myšlenky, že populace kořisti roste rychlostí úměrnou velikosti (neuvažujeme vnitrodruhovou konkurenci, to je realistické například pokud predátor udržuje velikost populace pod nosnou kapacitou prostředí) a kořist je působením predátora odebírána rychlostí úměrnou velikosti populace dravce a populace kořisti (více dravců více omezí růst populace a dravci jsou v lovu úspěšnější, pokud je kořisti více). Populace predátora bez přítomnosti kořisti klesá rychlostí úměrnou velikosti populace predátora a přítomnost kořisti se projeví v růstu populace predátora pozitivním faktorem. Podobně jako u kořisti, předpokládejme, že tento faktor je úměrný velikosti populace kořisti i predátora. Matematicky zapsáno, soužití kořisti a dravce je modelováno autonomním systémem

$$\frac{\mathrm{d}x}{\mathrm{d}t} = ax - bxy$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = -cy + dxy$$

Nápověda příkazu `solve_ivp` obsahuje řešení Lotkova–Volterrova modelu. Řešení jsou periodické funkce, za maximem kořisti následuje maximum dravce a pokles populace kořisti. Malý stav kořisti způsobí pokles populace dravce, který nemá dost potravy. To uvolní prostor pro růst populace kořisti. Hojnost kořisti ovšem podpoří populaci dravce, která začne se zpožděním růst kopírovat. Mnoho dravců ale hodně uloví a tím se růst populace kořisti zastaví a nastane pokles. Tím se cyklus uzavře.



Stacionární bod je řešením soustavy rovnic

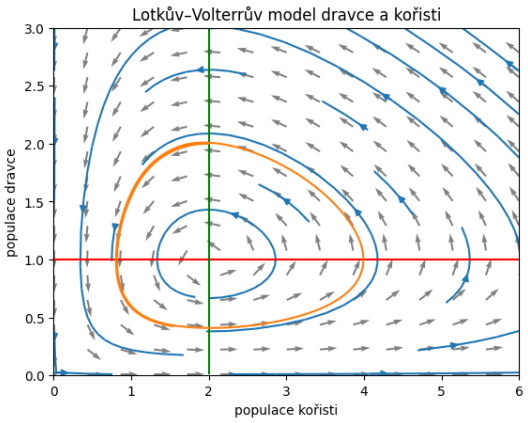
$$0 = ax - bxy$$
$$0 = -cy + dxy$$

a za přirozeného předpokladu $x > 0$ a $y > 0$ (obě populace jsou přítomny) je možné tento systém vydělit do tvaru

$$0 = a - by$$
$$0 = -c + dx,$$

odkud plyne $x = \frac{c}{d}$ a $y = \frac{a}{b}$. To je zajímavé. Stacionární stav kořisti je určen parametry růstu populace dravce a stacionární stav dravce je vyjádřen pomocí parametrů růstu populace kořisti.

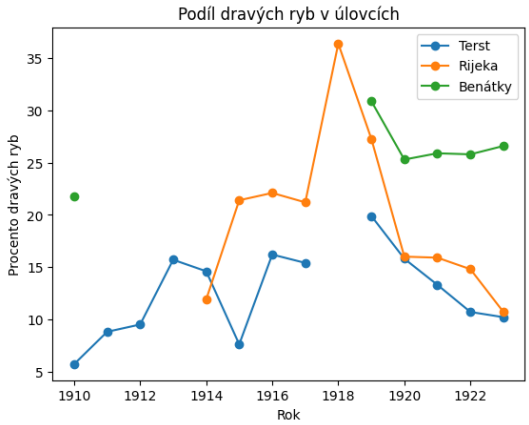
Ve fázovém prostoru vidíme, že řešení je cyklus. Pro více informací zakreslíme do fázového prostoru nejenom řešení, ale i nulkliny a směrové pole. Nulklinou kořisti (křivka kdy je růst kořisti nulový) je přímka $y = a/b$, tj. vodorovná přímka procházející stacionárním bodem. Každá trajektorie tuto přímku protíná svisle. Nulklinou dravce (křivka kdy je růst dravce nulový) je přímka $x = c/d$, tj. svislá přímka procházející stacionárním bodem. Každá trajektorie tuto přímku protíná vodorovně. Všechny trajektorie jsou uzavřené cykly. Na obrázku je několik částí těchto trajektorií vykreslených příkazem `streamplot` a jedna celá trajektorie ve tvaru cyklu.



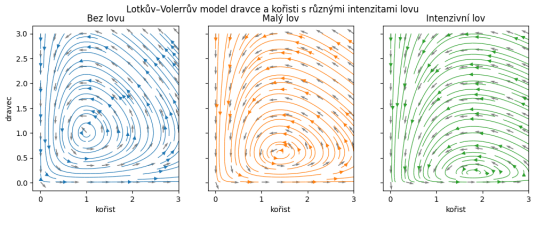
8.5 Volterrův efekt

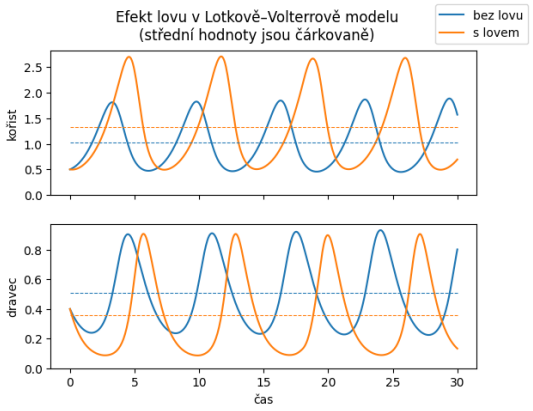
Na uvedeném modelu si můžeme demonstrovat Volterrův efekt. Toto je situace, kdy zásah proti škůdcům s cílem redukovat jejich populaci může v dlouhodobém měřítku paradoxně vyvolat přemnožení těchto škůdců. Například použití insekticidu nebo pesticidu může krátkodobé snížit populaci, kterou se snažíme redukovat, ale pokud jsou na použitou látku citliví i predátoři, živící se tímto škůdcem, dojde k výrazné redukci i u populace predátorů. To znamená, že škůdce ztratí svého přirozeného nepřítele a tím je mu umožněn nejen nárůst na původní hodnoty, ale i přemnožení. V konečném důsledku dojde k přemnožení populace škůdce, jehož počty jsme postřikem chtěli redukovat

Volterrův efekt byl vlastně i motivací pro sestavení modelu. Na tento efekt a zajímavé zvýšení procenta dravých ryb v sítích po omezení rybolovu během první světové války Volterru upozornil jeho zeť, biolog Umberto d'Ancona.



Budeme základní model modifikovat tak, že do populace kořisti i dravce přidáme lov s konstantním úsilím, tj. úměrný lovené populaci. Například pro lov ryb to věrně modeluje situaci, že procento dravých ryb uvízlých v rybářských sítích roste s tím, jak roste procento dravých ryb v mořích celkově.




Efekt lovu v Lotkově-Volterrově modelu
(střední hodnoty jsou čárkovaně)
bez lovu
s lovem
kořist
dravec
čas
0.0
0.5
1.0
1.5
2.0
2.5
0.2
0.4
0.6
0.8
0
5
10
15
20
25
30

KAPITOLA 9

Modely konkurence populací

9.1 Konkurence dvou populací

Předpokládejme, že v jisté izolované oblasti jsou přítomny dva živočišné druhy, které si vzájemně konkurují. Předpokládejme dále, že konkurence ovlivní rychlost růstu daného druhu tak, že zpomalení rychlosti růstu je přímo úměrné četnosti, s níž se jedinci jednoho druhu setkávají s jedinci druhého druhu. Nepředpokládáme přitom nic bližšího o typu této konkurence, tj. zda jedinci jednoho druhu fyzicky brání jedincům druhého druhu v přístupu k potravě, či zda je konkurence založena jenom na tom, že „ujídají ze společného krajíce“. Náš model bude zahrnovat oba typy konkurence, změna se projeví pouze ve velikosti konstanty úměrnosti, která vyjadřuje zpomalení vývoje populací vlivem konkurence.

9.1.1 Matematický model

Je-li velikost první populace vyjádřena veličinou x a velikost druhé populace veličinou y, je možno systém popsat soustavou diferenciálních rovnic

$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= (a - bx)x - cxy,\\ \frac{\mathrm{d}y}{\mathrm{d}t} &= (\alpha - \beta y)y - \gamma xy,\end{aligned}$$

kde všechno kromě x,y jsou kladné reálné parametry.

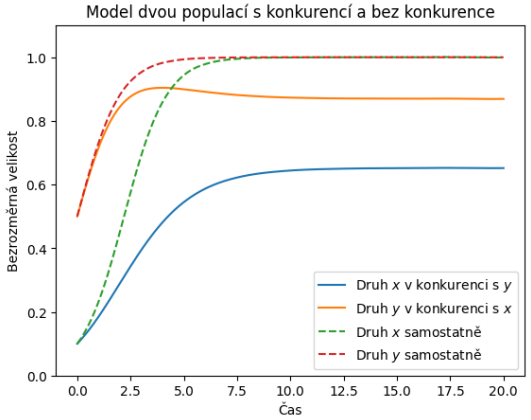
Prozkoumejme strukturu tohoto systému. První členy na pravých stranách soustavy odpovídají logistickému růstu populace, pokud se tato populace nachází v prostředí osamocena. Druhé členy obsahují velikosti obou populací a odpovídají zpomalení růstu vlivem mezidruhové konkurence.

Frekvence setkávání jedinců populace x s jedinci populace y je úměrná součinu velikostí těchto populací xy a zpomalení růstu populace x vlivem těchto setkání je úměrné tomuto součinu s konstantou úměrnosti c. Parametr c tedy reprezentuje sílu konkurence, tj. jaký vliv má vzájemná konkurence na růst populace x. Vliv populace x na růst populace y je dán parametrem γ, který obecně může být různý od c.

Prozkoumejme ještě možná jednodruhová podspolečenstva. Pro $y = 0$ je přítomna pouze populace x. První rovnice pak představuje logistickou rovnici pro vývoj populace x. Nosná kapacita prostředí je $\frac{a}{b}$.

Podobně, je-li $x = 0$, představuje druhá rovnice v soustavě logistickou rovnici pro růst populace y s nosnou kapacitou prostředí $\frac{\alpha}{\beta}$.

Řešení pro konkrétní sadu parametrů a konkrétní sadu počátečních podmínek najdeme stejně snadno, jako řešení jedné diferenciální rovnice. Vykreslíme si pro srovnání čárkovaně do stejného grafu i dynamiku samostatných jednodruhových populací.



9.1.2 Fázový portrét

Prozkoumáme fázový portrét. Řešení tedy budeme uvažovat jako paametrické křivky v rovině, kde na každé ose je velikost jedné z populací.

Pro nalezení nulklin přepíšeme soustavu do tvaru

$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= (a - bx - cy)x,\\ \frac{\mathrm{d}y}{\mathrm{d}t} &= (\alpha - \beta y - \gamma x)y.\end{aligned}$$

Systém má dvě x-nulkliny

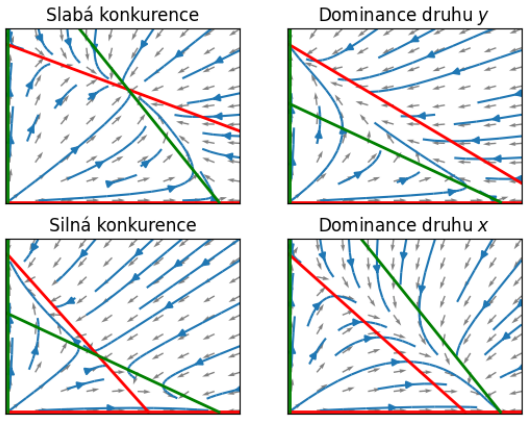
$$\begin{aligned}n_{1x}: &\quad x = 0,\\ n_{2x}: &\quad a - bx - cy = 0,\end{aligned}$$

a dvě y-nulkliny

$$\begin{aligned}n_{1y}: &\quad y = 0,\\ n_{2y}: &\quad \alpha - \beta x - \gamma y = 0.\end{aligned}$$

V průsečících x-nulklin a y-nulklinami nalezneme stacionární body systému.

Nulklina n_{2x} je přímka procházející body $\left[0, \frac{a}{c}\right]$ a $\left[\frac{a}{b}, 0\right]$. Nulklina n_{2y} je přímka procházející body $\left[\frac{\alpha}{\gamma}, 0\right]$ a $\left[0, \frac{\alpha}{\beta}\right]$. Podle vzájemné polohy těchto bodů ma osách x a y jsou možné čtyři kvalitativně odlišné případy znázorněné na obrázcích. Zeleně jsou znázorněny x-nulkliny a červeně y-nulkliny. Stacionární body jsou v průsečíku nulklin různých barev.



9.1.3 Analýza stacionárních bodů

Prozkoumáme chování trajektorií v okolí stacionárních bodů.

Jacobiho matice systému je

$$J(x, y) = \begin{pmatrix} a - 2bx - cy & -cx \\ -\gamma y & \alpha - 2\beta y - \gamma x \end{pmatrix}.$$

Označme specifické míry růstu populací (per capita) $m(x, y) = a - bx - cy$ a $\mu(x, y) = \alpha - \beta y - \gamma x$.

Stacionární bod v počátku

Průsečíkem nulklin n_{1x} a n_{1y} je stacionární bod $S_1 = [0, 0]$. Tento bod odpovídá stavu, kdy v prostředí není přítomna žádná z populací x, y. Pronikne-li do takového prostředí malé množství jedinců populace x, začnou se množit rychlostí $m(0, 0) = a$. Invazní parametr je tedy kladný a populace se uchytí. Podobně invazní parametr populace y je $\mu(0, 0) = \alpha > 0$ a i populace y se v prostředí uchytí a začne množit. Jacobiho matice v bodě S_1 je

$$J(0, 0) = \begin{pmatrix} a & 0 \\ 0 & \alpha \end{pmatrix}$$

a protože $\det J(0, 0) = \alpha a > 0$ a $\mathrm{Tr}\, J(0, 0) = a + \alpha$, je v bodě S_1 nestabilní uzel nebo nestabilní ohnisko. Vzhledem k tomu, že žádná trajektorie nemůže vystoupit z prvního kvadrantu (vysvětlete proč!), jedná se o nestabilní uzel.

Stacionární body na osách

Průsečíkem nulklin n_{1x} a n_{2y} je stacionární bod $S_2 = [0, \frac{\alpha}{\beta}]$. Tento stav odpovídá tomu, že populace x se v prostředí nevyskytuje a velikost populace y je ustálena na hodnotě nosné kapacity prostředí. Jakákoliv náhodná změna populace y vede k tomu, že se systém vrátí opět do tohoto stacionárního bodu, jak víme ze studia logistické rovnice. Prozkoumejme, jak se systém chová při malých náhodných změnách populace x. Invazní parametr této populace do stavu S_2 je $m\left(0, \frac{\alpha}{\beta}\right) = a - c\frac{\alpha}{\beta} = c\left(\frac{a}{c} - \frac{\alpha}{\beta}\right)$. Tento výraz je kladný pokud

$$\frac{a}{c} > \frac{\alpha}{\beta}, \tag{9.1}$$

tj. pokud parametr c, charakterizující sílu mezidruhové konkurence, je dostatečně malý. V tomto případě se populace x uchytí a začne množit. Platí-li opačná nerovnost, bude invazní parametr populace x záporný, populace bude vymírat a systém bude dospívat zpět do stacionárního stavu S_2. Jacobiho matice v bodě S_2 je

$$J\left(0, \frac{\alpha}{\beta}\right) = \begin{pmatrix} a - c\frac{\alpha}{\beta} & 0 \\ \gamma\frac{\alpha}{\beta} & -\alpha \end{pmatrix}$$

a protože $\det J(0, \frac{\alpha}{\beta}) = -ac\left(\frac{a}{c} - \frac{\alpha}{\beta}\right)$, je v bodě S_2 sedlo, jestliže platí (9.1) a uzel nebo ohnisko jinak. Vzhledem k tomu, že trajektorie nemůže protnout osu y (proč?), nejedná se o ohnisko. Vzhledem k tvaru směrového pole se tedy jedná sedlo nebo o stabilní uzel, podle toho, zda je splněna podmínka (9.1) či nikoliv.

Stacionární bod uvnitř prvního kvadrantu

Průsečíkem nulklin n_{2x} a n_{1y} je stacionární bod $S_3 = [\frac{a}{b}, 0]$. Situace je analogická jako u stacionárního bodu S_2, pouze jsou vyměněny role populací x a y. Systém tedy v nepřítomnosti populace y dospěje do stavu, kdy populace x se ustálí na hodnotě nosné kapacity prostředí, invazní parametr populace y do tohoto stavu je $\mu(\frac{a}{b}, 0) = \gamma\left(\frac{\alpha}{\gamma} - \frac{a}{c}\right)$ a je kladný pokud

$$\frac{\alpha}{\gamma} > \frac{a}{b} \tag{9.2}$$

a záporný jinak. Bod S_3 je sedlo, pokud uvedená podmínka platí a stabilní uzel jinak.

Průsečík nulklin n_{2x} a n_{2y} získáme řešením soustavy rovnic

$$\begin{aligned} a - bx - cy &= 0, \\ \alpha - \beta y - \gamma x &= 0. \end{aligned} \tag{9.3}$$

Toto lze provést numericky, přímým řešením, nebo graficky. Zajímáme se přitom pouze o řešení, které se nachází v prvním kvadrantu, tj. jehož obě složky jsou nezáporné. Z náčrtů je patrné, že přímky odpovídající těmto nulklinám se protnou v prvním kvadrantu právě tehdy, když buď platí obě z nerovností (9.1), (9.2), nebo ani jedna. Označme průsečík nulklin $S_4 = (x^*, y^*)$. I když hodnoty x^*, y^* je možno explicitně vypočítat, uvidíme, že jejich znalost nebude pro vyšetřování typu stacionárního bodu důležitá. Jacobiho matice v bodě S_4 je

$$J(x^*, y^*) = \begin{pmatrix} a - 2bx^* - cy^* & -cx^* \\ -\gamma y^* & \alpha - 2\beta y^* - \gamma x^* \end{pmatrix}.$$

Vzhledem k tomu, že (x^*, y^*) jsou řešením (9.3), platí

$$\begin{aligned} a - bx^* - cy^* &= 0, \\ \alpha - \beta y^* - \gamma x^* &= 0 \end{aligned}$$

a proto

$$J(x^*, y^*) = \begin{pmatrix} -bx^* & -cx^* \\ -\gamma y^* & -\beta y^* \end{pmatrix}.$$

Stopa Jacobiho matice je vždy záporná. Determinant Jacobiho matice je

$$\det J(x^*, y^*) = b\beta x^* y^* - c\gamma x^* y^* = x^* y^* (b\beta - c\gamma).$$

Pokud platí (9.1) a (9.2), pak

$$b\beta > \frac{\alpha c}{a}\frac{a\gamma}{\alpha} = c\gamma,$$

determinant Jacobiho matice je kladný a bod S_4 je ohnisko nebo uzel. Vzhledem k tvaru směrového pole se nemůže jednat o ohnisko a bod je tedy stabilní uzel. Pokud neplatí ani jedna z nerovností (9.1), (9.2), je determinant Jacobiho matice naopak záporný a v bodě S_4 je sedlo.

Shrnutí

- Platí-li obě z nerovností (9.1), (9.2), leží uvnitř prvního kvadrantu singulární bod, který je stabilním uzlem. Všechny trajektorie ležící v prvním kvadrantu konvergují do tohoto bodu pro $t \to \infty$. Systém tedy vždy dospěje do stavu, kdy přežívají obě populace. Podmínky (9.1) a (9.2) jsou splněny, pokud jsou parametry c, γ, které charakterizují mezidruhovou konkurenci, dostatečně malé. Proto tento stav budeme nazývat *slabou konkurencí*.
- Platí-li (9.1) a neplatí-li (9.2) nemá systém uvnitř prvního kvadrantu singulární bod a všechny trajektorie ležící uvnitř tohoto kvadrantu konvergují pro $t \to \infty$ do stacionárního bodu $[\frac{a}{b}, 0]$. V systému tedy dojde vlivem mezidruhové konkurence k vyloučení druhu y a populace druhu x se ustálí na hodnotě, odpovídající jeho nosné kapacitě prostředí. Tento stav budeme nazývat *dominancí druhu x*.
- Platí-li (9.2) a neplatí-li (9.1), je situace podobná jako v předchozím bodě, pouze dochází ke konkurenčnímu vyloučení druhu x a jedná se tedy o *dominanci druhu y*.
- Neplatí-li ani jedna z nerovností (9.1), (9.2), má systém uvnitř prvního kvadrantu singulární bod, který je sedlem a tedy pouze konečný počet trajektorií konverguje k tomuto bodu. Tento bod však nebude odpovídat reálnému stavu, ve kterém se může populace trvale nacházet, protože není odolný vůči náhodným perturbacím. Trajektorie, konvergující do tohoto sedlového bodu, rozdělí první kvadrant na dvě množiny, které tvoří oblasti atraktivity jednotlivých stabilních stacionárních bodů. Všechny trajektorie tedy směřují pro $t \to \infty$ buď do stacionárního bodu $[\frac{a}{b}, 0]$, nebo $[0, \frac{\alpha}{\beta}]$. V systému tedy vždy dojde ke konkurenčnímu vyloučení jednoho z druhů. Který z druhů bude vyloučen, záleží na počátečních podmínkách. Pokud jsou počáteční podmínky nastaveny tak, že trajektorie začíná v oblasti atraktivity stacionárního bodu $[\frac{a}{b}, 0]$, dojde k eliminaci druhu y, pokud je tomu naopak, dojde k eliminaci druhu x. Nesplnění nerovností (9.1), (9.2) odpovídá tomu, že parametry mezidruhové konkurence jsou dostatečně velké, proto tento stav budeme nazývat *silnou konkurencí*.

Všechny typy konkurence (konkurenční vyloučení, slabá konkurence, silná konkurence) jsou v přírodě pozorovány. V ekologii zpravidla největší pozornost upoutává slabá konkurence, kdy dochází ke stabilní koexistenci. Tento jev nastává, pokud, řečeno v biologických termínech, jedinec daného druhu svou existencí konkuruje jedincům svého druhu více, než jedincům druhu jiného. V praxi to znamená, že druhy musí mít poněkud odlišné ekologické nároky (Gauseho princip). Například společně hnízdící druhy ptáků si konkurují v boji o hnízdiště. Tyto druhy mohou koexistovat, pokud mají například rozdílné slo-

žení potravy. V tomto případě jedinec konkuruje svému druhu i v boji o prostor k hnízdění i v boji o potravu, jedincům druhého druhu však konkuruje méně – pouze v boji o hnízdiště. Všimněme si ještě, že pokud vedle sebe koexistují dvě populace, součet velikostí je větší než velikost rovnovážných stavů kterékoliv osamocené populace. Dvě konkurující-si populace tedy využívají zdroje efektivněji než populace jediná.

Lze ukázat, že ke konkurenčnímu vyloučení slabšího druhu nemusí dojít, pokud druhy žijí v komplikovanějších podmínkách, než jaké jsme dosud uvažovali: například v přítomnosti třetího konkurenta, v přítomnosti predátora, v periodicky se měnícím životním prostředí, či pokud na sebe druhy reagují se zpožděním, ve fragmentovaném prostředí, kdy silnější druh neobsadí všechny fragmenty a zůstane tak volný prostor pro slabší druh.

9.2 Model konkurence tří druhů

Podobný model, jako pro konkurenci dvou druhů, je možno sestavit i pro konkurenci tří a více druhů. Matematický popis je v tomto případě poněkud obtížnější, proto se omezíme na formulaci systému diferenciálních rovnic a na některé výsledky.

Označíme-li velikost populace jednotlivých druhů veličinami x, y, z, je možno model konkurence mezi těmito druhy zapsat ve tvaru

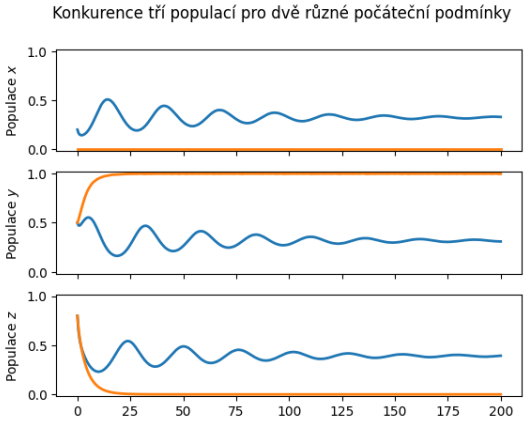
$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= (a - bx - cy - dz)x,\\ \frac{\mathrm{d}y}{\mathrm{d}t} &= (\alpha - \beta x - \gamma y - \delta z)y,\\ \frac{\mathrm{d}z}{\mathrm{d}t} &= (m - nx - oy - pz)z,\end{aligned}$$

kde všechny veličiny kromě x, y, z jsou kladné parametry.

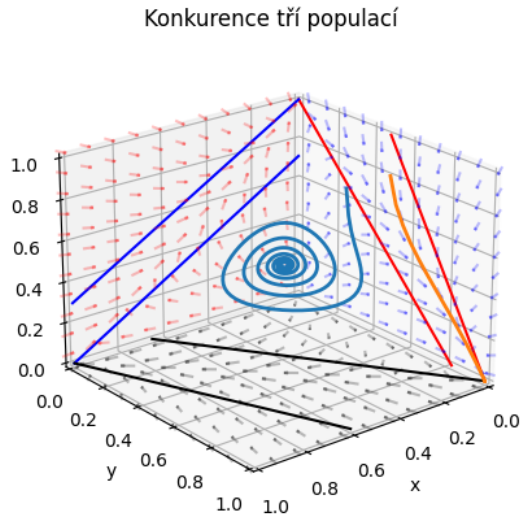
Je-li v lokalitě přítomen pouze jeden druh (např. x), vyvíjí se podle logistické rovnice a velikost populace se ustálí na hodnotě dané úživností prostředí. O tom, zda do takového stavu mohou proniknout druhy z nebo y a úspěšně se začít rozmnožovat rozhodují znaménka invazních parametrů těchto populací v uvažovaném stacionárním bodě. Například populace y má ve stacionárním bodě $[\frac{a}{b}, 0, 0]$ invazní parametr $\alpha - \beta\frac{a}{b} - \gamma 0 - \delta 0 = \beta\left(\frac{\alpha}{\beta} - \frac{a}{b}\right)$ a populace z má v tomtéž bodě invazní parametr $n\left(\frac{m}{n} - \frac{a}{b}\right)$.

Není-li druh z ve skutečnosti v dané lokalitě přítomen, je $z = 0$ a první dvě rovnice zcela odpovídají modelu konkurence mezi dvěma populacemi x a y. Podobně je možno studovat další dvojdruhová společenství x,z a y,z. Pokud invazní parametr každého druhu, který není ve společenstvu přítomen, do stavu reprezentovaného stabilními stavy dalších dvou druhů, je kladný, systém je permanentní a velikost všech populací se udrží na nenulové hodnotě. Tato podmínka kladnosti všech invazních parametrů je však pouze dostatečná, nikoliv nutná.

Numerickým řešením tohoto systému je možné ukázat, že při vhodném nastavení parametrů mezidruhové konkurence existuje situace, kdy libovolné dvojdruhové podspolečenství druhů x,y,z není permanentní a dojde ke konkurenčnímu vyloučení některého druhu a k obsazení dané lokality jediným druhem, zatímco v přítomnosti nenulového stavu všech tří populací existuje ustálený stav, charakterizující přežívání všech tří populací. V takovém případě se tedy, poněkud paradoxně, kromě konkurence jedná i o jistý druh symbiózy. Je možné, že pokud jeden z druhů náhodnými vlivy nebo neopatrným zásahem člověka vymře, nemusí toto vymření vést k posílení populací jeho konkurentů, ale někteří z konkurentů mohou vyhynout, zatímco jiní dominují.



Na obrázku je schematicky zachycena jedna z takových situací. V průmětnách jsou zachyceny poměry pro jednotlivá dvojdruhová společenství. Odsud plyne, že v případě dvojdruhového společenství zy (červená rovina) dominuje druh y, v případě společenství xz (modrá rovina) dominuje druh z a u společenství xy (černá rovina) dominuje druh x. Na osách jsou vyznačeny stacionární body charakterizující nosnou kapacitu prostředí pro každou z populací, pokud je populace v prostředí sama. Průsečíky zaznačených přímek s osou x jsou (v pořadí od počátku) $\frac{\alpha}{\beta}$, $\frac{a}{b}$ a $\frac{m}{n}$, s osou y $\frac{m}{o}$, $\frac{\alpha}{\gamma}$ a $\frac{a}{c}$ a s osou z body $\frac{a}{d}$, $\frac{m}{p}$ a $\frac{\alpha}{\delta}$.



Teto příklad ukazuje, že v lokalitě obsazené větším počtem druhů, je možná koexistence druhů, které jinak koexistovat nemohou. *V ekologii se zpravidla druhově rozmanitější společenství považují za stabilnější a odolnější vůči jakýmkoliv změnám.*

9.3 Symbióza dvou druhů

Podobně jako jsme modelovali konkurenci dvou druhů, můžeme modelovat i symbiózu dvou druhů – situaci, kdy přítomnost jedné populace podporuje růst populace druhé. tento stav lze popsat soustavou diferenciálních rovnic obdobnou soustavě pro konkurenci, pouze změníme znaménka u členů charakterizujících vnitrodruhové působení. Studujme tedy soustavu

$$\begin{aligned} x' &= (a - bx)x + cxy, \\ y' &= (\alpha - \beta y)y + \gamma xy, \end{aligned}$$

kde x, y jsou velikosti jednotlivých populací a ostatní veličiny jsou kladné reálné parametry. Podobná analýza jako u modelu konkurence dvou druhů přináší následující výsledky.

- Tři stacionární body odpovídají jednomu stavu bez přítomnosti obou populací, a dvěma stavům, kdy v prostředí je přítomna pouze jedna z populací, jejíž velikost je ustálena na hodnotě kapacity prostředí. Tyto stacionární body jsou nestabilní.
- Jestliže míra symbiózy je dostatečně silná; přesněji, pokud platí

  $$c\gamma > b\beta,$$

  pak systém nemá žádný další stacionární bod a všechny trajektorie jsou pro $t \to \infty$ neohraničené. Tento stav odpovídá situaci, kdy míra užitku ze symbiózy převáží vliv vnitrodruhové konkurence, která jinak zastavuje růst. Protože růst obou populací není ohraničený, obě populace po přemnožení zničí své životní prostředí. V angl. literatuře charakterizuje tento stav termín „orgy of mutual beneficion“ (orgie vzájemné dobročinnosti).
- Jestliže míra symbiózy je malá, tj. jestliže platí

  $$c\gamma < b\beta,$$

  má systém stabilní stacionární bod v prvním kvadrantu. Velikosti populací se ustálí na hodnotě, odpovídající tomuto bodu. Velikost obou populací bude převyšovat úživnou hodnotu prostředí každé z populací, populace tedy bude mít větší velikost, než jakou by měla, kdyby se v prostředí nacházela osamocena.

Další pojednání o modelech konkurence je možno nalézt například v [3] kapitola 7.4 a [15] str. 220.

KAPITOLA 10

Modely dravce a kořisti

Modely [3] kapitola 10

KAPITOLA 11

Diskrétní populační modely

Diskrétní populační modely jsou zpravidla rekurentní vzorce, které určují velikost populace nebo populací v dalším roce na základě velikostí v předchozím roce (modely prvního řádu) nebo několika předchozích let (modely vyššího řádu). Je-li závislost lineární, můžeme tyto modely zapsat pomocí maticového násobení, jak jsme poznali již dříve například u Leslieho matice. To je výhodné, protože můžeme využít rozvinutý aparát lineární algebry.

Výhodou diskrétních maticových modelů oproti spojitým je jednodušší řešení. To spočívá v opakovaném používání modelu a v postupném výpočtu populačních stavů jenom použitím jednoduchých matematických operací. Nevýhodou je horší možnost nalezení analytického řešení a že se modely někdy mohou chovat nečekaně, viz například chaos v diskrétní logistické rovnici.

11.1 Diskrétní modely jednodruhové populace

11.1.1 Základní diskrétní modely

Diskrétní model jednodruhové populace modelovaný rovnicí prvního řádu má obecný tvar

$$x(k+1) = x(k)f(x(k)),$$

kde funkce f charakterizuje působení vnitrodruhové konkurence.

Základní diskrétní model, diskrétní logistickou rovnici, jsme poznali dříve. Tento model je jednoduchý, ale má komplikované chování, které může vést k periodickým řešením s vysokou periodou a k chaosu.

Ukážeme si volbu tří funkcí, které mají společné chování v počátku a při dosažení nosné kapacity prostředí. Konkrétně, platí $f(0) = r$ a $f(K) = 1$.

- Logistický růst

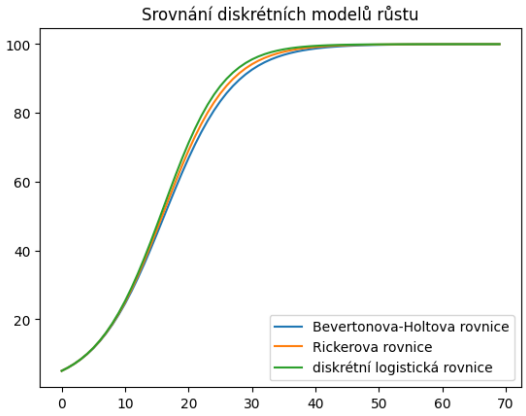
$$x(k+1) = rx(k)\left(1 - \frac{x(k)}{K}\frac{r-1}{r}\right)$$

- Bevertonova–Holtova rovnice, logistická rovnice Pielou:

$$x(k+1) = x(t)\frac{rK}{K + (r-1)x(k)}$$

- Rickerova logistická rovnice:

$$x(k+1) = x(k)e^{\left(1-\frac{x(t)}{K}\right)\ln r}$$



11.1.2 Periodické cikády

Následující model je zpracován podle [13] a vysvětluje zajímavý efekt vyskytující se v životě cikád *Magicicada septendecim, M. cassini* a *M. septendecula*.

Dospělé cikády během pár týdnů svého života nakladou vajíčka u stromů, kde se vylíhly. Z vajíček se vyklubou larvy, které se zavrtají pod zem ke kořenům. Životní cyklus cikád se liší od 3 do 4, 7, 13 nebo 17 let. Poslední dva typy cikád se vyznačují (na východním pobřeží severní Ameriky) masovým synchronizovaným výskytem.

Předpokládejme, že druh má životní cyklus k let, larvy dospějí a objeví se na povrchu k let po smrti rodičů. Nechť n_t je počet larev, které se v roce t zavrtají do země, najdou útočiště a potravu u kořenů stromů a za k let se z nich (pokud přežijí) vylíhnou dospělé cikády.

Předpokládejme, jenom zlomek μ larev přežije do každého dalšího roku. V roce t se vylíhnou larvy, které se před k lety zavrtaly do země (jejich počet byl n_{t-k}) a v zemi přežily celkem k let (každý rok jich přežije jenom μ-násobek a proto k let přežije $\mu^k n_{t-k}$ larev). Tyto larvy se objeví v roce t jako dospělí, kteří uzavřou cyklus. Z tohoto počtu predátoři zlikvidují p_t cikád a pro další rozmnožování zůstane $[\mu^k n_{t-k} - p_t]_+$ jedinců, kde $[x]_+ = \max(x, 0)$. Jestliže se každé cikádě narodí průměrně f larev, přibude v tomto roce

$$M_t = f \cdot [\mu^k n_{t-k} - p_t]_+$$

larev.

Je-li D celková nosná kapacita prostředí, je *volná* kapacita dána vztahem

$$c_t = \left[D - \sum_{i=1}^{k-1} \mu^i n_{t-i} \right]_+ ,$$

protože část této kapacity obsadily larvy z loni, předloni, předpředloni atd. Z M_t narozených larev se maximálně c_t může zavrtat do země a přežít. Proto počet larev v dalším roce dán vztahem $n_t = \min(M_t, c_t)$.

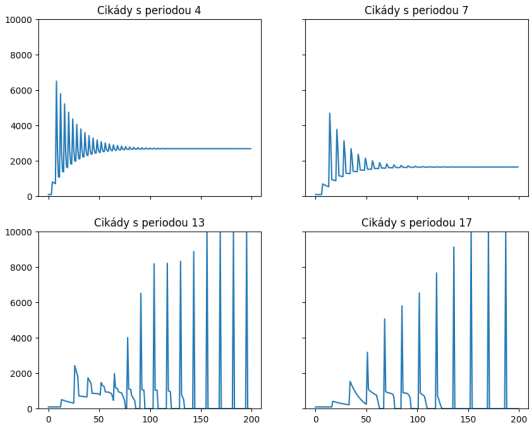
Působení predátorů (živí se dospělými cikádami) můžeme modelovat rovnicí

$$p_t = \nu p_{t-1} + a\mu^k n_{t-k-1},$$

kde $\nu < 1$ (bez cikád predátoři vymírají, toto vymírání se zastavuje s počtem cikád, které se loni vylíhly).

Pro lepší formulaci numerického modelu posuneme index t o jedničku.

$$\begin{aligned} p_{t+1} &= \nu p_t + a\mu^k n_{t-k} \\ c_{t+1} &= \left[D - \sum_{i=1}^{k-1} \mu^i n_{t+1-i} \right]_+ \\ M_{t+1} &= f \cdot [\mu^k n_{t+1-k} - p_{t+1}]_+ \\ n_{t+1} &= \min(M_{t+1}, c_{t+1}) \end{aligned}$$



11.2 Diskrétní modely dvou populací

KAPITOLA 12

Difuzní rovnice

Část II

Cvičení

KAPITOLA 13

První kroky

13.1 Dva druhy políček

V zápisníku jsou dva druhy políček. Políčko s textem a políčko s kódem jazyka Python.

- Toto políčko je políčko s textem.
- Formátování pomocí jazyka Markdown.
- Je možné vkládat matematické výrazy. Rovnice

$$ax^2 + bx + c = 0$$

s neznámou x má řešení

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

- Je možné vkládát odkazy a obrázky.
- Políčko přepněte do editačního módu kliknutím myší nebo stiskem Enter.
- Připište sem co chcete a uložte stisknutím ctrl+Enter nebo shift+Enter.

Přidat text můžete jako další odstavec. Můžete přidat i další odrážku nebo začít psát jiný odrážkový seznam.

Odstavce jsou odděleny prázdným řádkem.

Následující políčko je políčko s příkazy jazyka Python.

1. Otevřete políčko.
2. Upravte podle potřeby.
3. Spusťte pomocí Shift+Enter.

```
1.2 * 6.56 - 10
```

```
-2.128000000000001
```

13.2 Knihovny

Některé funkce, konstanty a příkazy jsou přístupné v knihovnách. Knihovny musíme naimportovat a jejich název použít při volání.

```
import numpy as np
np.pi
```

```
3.141592653589793
```

```
# sinus praveho uhlu (uhel je␣
↪v~radianech)
np.sin(np.pi/2)
```

```
1.0
```

13.3 Cykly

Skriptování nám umožní spouštět příkazy opakovaně ta tak se vyhnout nudným činnostem. Základním cyklem je cyklus `for`, který je v následující ukázce. Více možností cyklu je v samostatné kapitole.

Třeba dokážeme vyřešit rovnici

$$x = \cos(x)$$

opakovaným dosazením do vzorečku.

```
x = 1
for i in range(50):
    x = np.cos(x)
    print(x)
```

```
0.5403023058681398
0.8575532158463934
0.6542897904977791
0.7934803587425656
0.7013687736227565
0.7639596829006542
0.7221024250267077
0.7504177617637605
0.7314040424225098
0.7442373549005569
0.7356047404363474
0.7414250866101092
0.7375068905132428
0.7401473355678757
0.7383692041223232
0.7395672022122561
0.7387603198742113
0.7393038923969059
0.7389377567153445
0.7391843997714936
0.7390182624274122
0.7391301765296711
0.7390547907469174
0.7391055719265363
0.7390713652989449
0.7390944073790913
0.739078885994992
0.7390893414033927
0.7390822985224024
0.7390870426953322
0.7390838469650002
0.7390859996481299
0.7390845495752126
0.7390855263619245
0.7390848683867142
0.7390853116067619
0.7390850130484203
0.7390852141609171
0.739085078689123
0.7390851699445544
0.7390851084737987
0.7390851498812394
0.7390851219886894
0.7390851407774467
0.7390851281211138
0.7390851366465718
0.7390851309037207
0.7390851347721744
0.7390851321663374
0.7390851339216605
```

Kód můžeme modifikovat. Například následující kód je podobný a vyřeší rovnici

$$x = x - \frac{x - \cos(x)}{1 + \sin(x)}.$$

Tato rovnice je sice ekvivalentní s předešlou rovnicí, ale čísla se ustálí mnohem rychleji.

```
x = 1
for i in range(10):
```

(continues on next page)

(pokračujte na předchozí stránce)

```
    x = x - (x-np.cos(x))/(1+np.sin(x))
    print(x)
```

```
0.7503638678402439
0.7391128909113617
0.739085133385284
0.7390851332151607
0.7390851332151607
0.7390851332151607
0.7390851332151607
0.7390851332151607
0.7390851332151607
0.7390851332151607
```

13.4 Často recyklujeme

Při zpracování dat nebo modelování dějů typicky řešíme stále stejné úlohy, jenom pro jiné situace. Proto typicky používáme stále podobné bloky příkazů, recyklujeme již napsaný kód. Často se používají hotové bloky příkazů (snippety), které se uživatel jenom upraví.

- V horním menu rozbale položku Snippets a vložte snippet „*Import knihoven*“. Ten zpřístupní knihovny pro numerické výpočty (už máme, ale to nevadí), kreslení grafů a práci s tabulkami. Jakmile snippet vložíte, blok příkazů spusťte.
- Poté vložte do dalšího políčka snippet „*Graf funkce, NumPy*“ a vykreslete si graf funkce

$$y = e^{-x^2}.$$

- Zkuste modifikovat příkazy tak, aby se kreslil graf kvadratické funkce, nebo sinusoida.

KAPITOLA 14

Úvod do jazyka Python

Python patří mezi nástroje pro dávkové zpracování dat. V tomto případě se nepostupuje interaktivně jako například v MS Excel, ale uživatel sestaví posloupnost příkazů a interpreter provede tyto příkazy podobně, jako spuštění programu. Výhody jsou následující.

- Možnost pracovat s daty neomezené velikosti.
- Snadná reprodukovatelnost.
- Lepší přehlednost v rozsáhlejších projektech.
- Možnost zařazení datové analýzy jako stavebního kamene delšího řetězce.
- Možnost neomezeného opakování stejné analýzy v cyklu nad jinými daty.

Nevýhoda dávkového zpracování je, že si nemůžeme požadované výstupy zajistit klikáním v menu. Jedná se však o nástroje s širokou uživatelskou základnou a v době kvalitního vyhledávání na Internetu není těžké „vygooglit“ na webech s dotazy a odpověďmi příkazy nutné pro kreslení grafů různých typů a další dílčí úkoly spojené se zpracováním dat. Výhodou jsou mnohem větší možnosti, než má Excel. Srovnatelný se skriptováním v jazyce Python je jedině program Matlab, který je na univerzitě dostupný, ale vzhledem k vysoké ceně nebývá dostupný mimo univerzity a mimo velké firmy.

14.1 Ke způsobu práce

- Učíme se na příkladech.
- Začínáme s modifikací hotových modelů. Experimentujte. Zkoušejte příklady z nápovědy, z přednášek a cvičení.
- Model stavíme z hotových bloků, málo modelů píšeme od nuly. Často najdeme podobný problém a ten si přizpůsobíme. Například najdeme obrázek podobný našemu zamýšlenému v galerii a odsud vidíme, jaké příkazy a volby je potřeba použít.
- Detaily k nastavení a volbám při volání funkcí hledáme v manuálu. Pro rychlou orientaci slouží cheatsheety.
- Spousta dotazů s kvalitními odpověďmi je na serveru StackExchange nebo jinde. Popište hesly problém ve vyhledávacím políčku Google a zkuste vyhledat podobné dotazy a odpovědi na ně. Totéž s chybovými hláškami, pokud jim nerozumíte.

14.2 K syntaxi v jazyce Python

- záleží na mezerách na začátku řádku
  - příkazy tvořící hlavní tělo začínají na začátku řádku
  - bloky uvnitř cyklů a podmínek jsou odsazeny o pevný počet mezer (doporučené jsou čtyři pro jednotlivé úrovně)
- zlom řádku může být uvnitř závorek s argumentem funkce a uvnitř závorek pro seznamy, potom nezáleží na odsazení
- komentáře jsou jednořádkové a uvozeny znakem #
- doporučený styl podle Style guide

```
pocet_opakovani = 10 # promenne␣
 ↪pojmenovavame tak, aby nazev␣
 ↪vypovidal o~obsahu dat
pocet_velkych_cisel = 0
"""
Při práci nemusíme vypisovat celé jméno␣
 ↪proměnné, ale můžeme používat
doplňování kódu, kdy napíšeme jenom␣
 ↪prvních několik písmen a po stisku
domluvené klávesy (zpravidla tabelátor)␣
 ↪se název buď doplní, nebo,
je-li více variant, se nabídnou možná␣
 ↪doplnění pomocí menu.
```

(continues on next page)

(pokračujte na předchozí stránce)

```
V tomto textu též vidíte možnost␣
 ↪víceřádkových komentářů. Stačí je
napsat jako víceřádkové řetězce, tj.␣
 ↪řetězce uvozené a ukončené třemi
uvozovkami.
"""
for i in range(pocet_opakovani):
    if i<5:
        # tisk informace o~cisle i
        print("Číslo",i,"je malé")
    else:
        print(
            "Číslo",
            i,
            "je velké"
            )
        pocet_velkych_cisel = pocet_
 ↪velkych_cisel + 1
print(
    "Vyskytlo se celkem",
    pocet_velkych_cisel,
    "velkych cisel"
)
```

```
Číslo 0 je malé
Číslo 1 je malé
Číslo 2 je malé
Číslo 3 je malé
Číslo 4 je malé
Číslo 5 je velké
Číslo 6 je velké
Číslo 7 je velké
Číslo 8 je velké
Číslo 9 je velké
Vyskytlo se celkem 5 velkych cisel
```

Všimněte si mimo jiné, že Python počítá a indexuje od nuly, podobně jako JavaScript a některé další jazyky. První položka v seznamu má index nula, druhá jedna atd.

14.3 Aritmetika s čísly

```
a = 4 # promenna a obsahuje nastavenou␣
 ↪hodnotu
b = a + 2  # promenna b bude o~dve␣
 ↪vetsi nez promenna a
a = b**2   # promenna a se nahradi␣
 ↪druhou mocninnou promenne b, promenna␣
 ↪b zustava na sve hodnote
a  # tisk hodnoty promenne, vystup␣
 ↪posledniho vypoctu se tiskne, neni␣
 ↪nutne pouzivat print
```

```
36
```

Proměnné si udrží hodnoty i v dalších políčkách. V dalším políčku zmenšíme hodnotu a o 30.

```
a-30
```

```
6
```

Operace se zapisují běžným způsobem jako například v Excelu, pouze umocňování se označuje dvojicí hvězdiček. Takto vypadá druhá mocnina trojky.

```
3**2
```

```
9
```

14.3.1 Úkol 1

Následující políčko s kódem `1**2 + 2**2 + 3**2 + 4**2` sčítá druhé mocniny přirozených čísel až do čísla 4. Opravte políčko tak, aby sečetlo druhou mocninu přirozených čísel až do čísla 8.

```
1**2 + 2**2 + 3**2 + 4**2
```

```
30
```

14.4 Hrátky s textem

Textový řetezec se bere jako seznam znaků a je možné ho ukládat do proměnná, přistupovat k prvnímu nebo poslednímu znaku, k několika prvním nebo několika posledním znakům, ke skupině znaků uprostřed atd. Pozor na to, že Python indexuje od nuly a první znak má tedy index nulový. Pokud je index záporný, znamená to pořadí od konce.

```
retezec="MENDELU"
retezec
```

```
'MENDELU'
```

```
retezec[0]
```

```
'M'
```

```
retezec[-2]
```

```
'L'
```

```
retezec[:4]
```

```
'MEND'
```

```
retezec[-3:]
```

```
'ELU'
```

```
retezec[2:4]
```

```
'ND'
```

```
retezec + " je prostě " + retezec + "."
```

```
'MENDELU je prostě MENDELU.'
```

```
veta = "".join([retezec," je prostě ",
 ↪retezec,"."])
veta
```

```
'MENDELU je prostě MENDELU.'
```

```
len(veta)
```

```
26
```

```
veta[:-5]
```

```
'MENDELU je prostě MEN'
```

```
(10*(retezec+"-"))[:-1]
```

```
'MENDELU-MENDELU-MENDELU-MENDELU-
 ↪MENDELU-MENDELU-MENDELU-MENDELU-
 ↪MENDELU-MENDELU'
```

```
"-".join([retezec for i in range(10)])
```

```
'MENDELU-MENDELU-MENDELU-MENDELU-
 ↪MENDELU-MENDELU-MENDELU-MENDELU-
 ↪MENDELU-MENDELU'
```

```
"-".join([retezec for i in range(10)]).
 ↪lower().replace("m","M")
```

```
'Mendelu-Mendelu-Mendelu-Mendelu-
 ↪Mendelu-Mendelu-Mendelu-Mendelu-
 ↪Mendelu-Mendelu'
```

Tyto techniky s přístupem k obsahu podle indexu využijeme, když budeme mít data v seznamu a budeme chtít přistupovat například k první nebo poslední hodnotě, nebo k několika prvním či několika posledním hodnotám.

14.4.1 Úkol 2

- Vložte pod toto políčko klávesou B (nebo pomocí menu) políčko pro vložení příkazů. Můžete také jít na následující políčko a vložit nové políčko klávesou A. Musíte být v příkazovém módu, tj. needitovat žádné pole. Kolem aktuálního políčka musí být modrý rámeček.
- Do proměnné `muj_pokus` uložte řetězec „LDF je nejlepší“.
- Určete délku řetězce pomocí funkce `len`.

14.5 Knihovny

Jenom základní funkce jsou v Pythonu přístupny přímo. Další funkce načítáme ve formě knihoven. Pro práci s daty zpravidla nejprve importujeme knihovny pro numeriku, práci s datovými tabulkami a pro kreslení grafů. Přitom používáme pro knihovny obvyklé zkratky, například `np` namísto `numpy`. Snipet pro vložení těchto knihoven najdete v rozbalovacím menu Snipets.

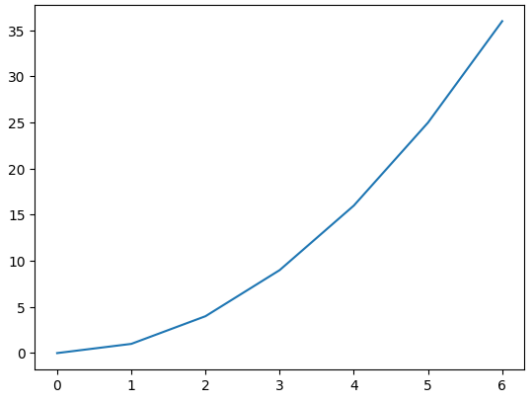
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

14.6 Práce se seznamem hodnot

14.6.1 Zadání hodnot přímo do seznamu

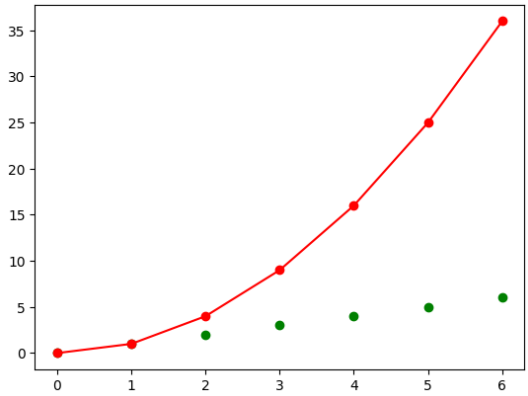
Pro nakreslení grafu vyzkoušejte následující kód.

```
x = [0, 1, 2, 3, 4, 5, 6]
y = [0, 1, 4, 9, 16, 25, 36]
plt.plot(x,y);
```



Grafy je možno modifikovat dle potřeby. Následující dvojice příkazů vykreslí dva grafy předvolenými barvami. Jeden graf je složen z teček, druhý z teček a lomené čáry.

```
plt.plot(x,x, "o", color="green")
plt.plot(x,y, "o-", color="red");
```



14.6.2 Vygenerování položek v seznamu

V předchozím jsme zadali funkční hodnoty přímo. Často se funkční hodnoty počítají. Například zde vypočteme druhé mocniny prvních několika přirozených čísel.

```
y = [i**2 for i in range(9)]
y
```

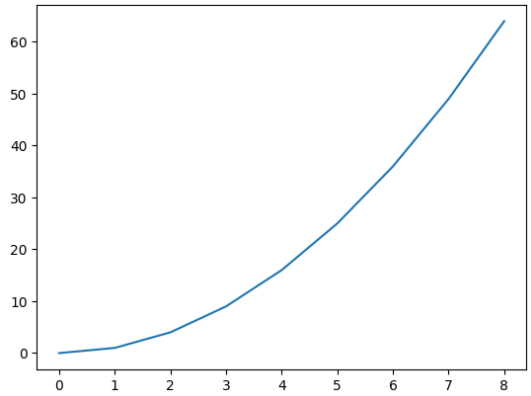
```
[0, 1, 4, 9, 16, 25, 36, 49, 64]
```

```
sum(y)
```

```
204
```

```
plt.plot(y)
```

```
[<matplotlib.lines.Line2D at␣
↪0x7f016bc12c50>]
```



Uvnitř hranatých závorek může být přechod na nový řádek. Na počtu mezer na začátku v tomto případě nezáleží.

> *Tuto techniku využijeme například, pokud budeme chtít vyřešit model pro různé parametry a výstupy uložit pro další práci.*

```
parametry = [0,1,2,4,6,15]
seznam = [
    i**3
    for i in parametry
]
seznam
```

```
[0, 1, 8, 64, 216, 3375]
```

14.7 Práce s poli `np.array`

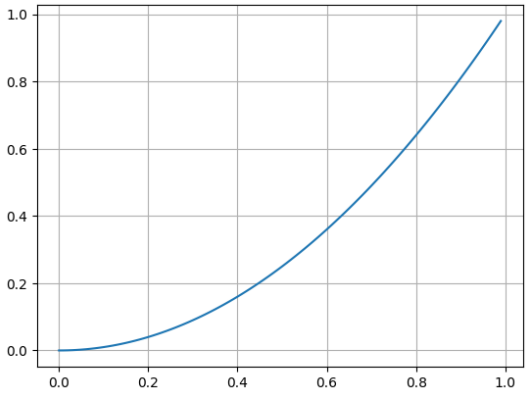
S poli typu `np.array` se pracuje podobně jako se seznamy ale dokážeme s nimi dělat rovnou matematické operace a zápis je pohodlnější.

```
a = [1,2,3,4,5]
A = np.array(a)

print(A)

for i in A[1:]:
    print("Prvek je "+str(i))

b = [10,20,30,40,50]
B = np.array(b)

#print(a*b)  # toto by zpusobilo chybu
print(A*B)
print(100*A+3)
print(A**2)
print(B**A)
print(10*a)
```

```
[1 2 3 4 5]
Prvek je 2
Prvek je 3
Prvek je 4
Prvek je 5
[ 10  40  90 160 250]
[103 203 303 403 503]
[ 1  4  9 16 25]
[       10       400     27000  ␣
 ↪2560000 312500000]
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2,␣
 ↪3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4,
 ↪ 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5,␣
 ↪1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2,
 ↪ 3, 4, 5]
```

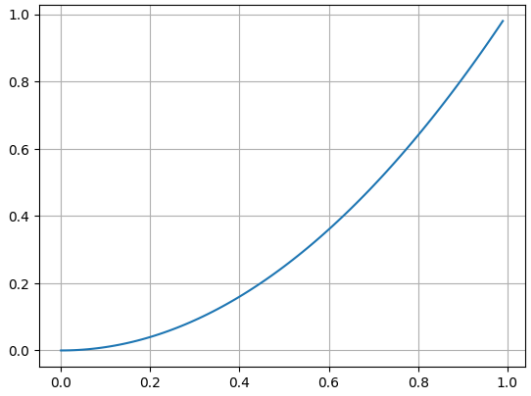
```
x = [0.01*i for i in range(100)]
y = [i**2 for i in x]
plt.plot(x,y)
plt.grid()
len(x)
```

```
100
```




```
# S~polem typu array je možné provést␣
 ↪operaci umocnění rovnou
N = 100
x = np.array([1/N*i for i in range(N)])
# jeste lepe by bylo nasledujici
# x = np.linspace(0,1,100)
y = x**2
plt.plot(x,y)
plt.grid()
len(x)
```

```
100
```



14.7.1 Úkol 3

Nakreslete graf třetí mocniny na intervalu $(-1, 1)$. Jako výchozí bod použijte snippet `Graf funkce, Numpy` (horní menu, rozbalovací položka `Snippets`).

```
# Sem vložte příkazy pro nakreslení␣
 ↪grafu funkce x**3 na intervalu od -1␣
 ↪do 1.
```

14.8 Tabulky, pandas

```
x = np.linspace(0,10,1001)
y = x**2
df = pd.DataFrame()
df["x"] = x
df["y"] = y
#df["suma"] = df["x"] + df["y"]
# Kratsi alternativa: df["suma"] = df.x␣
 ↪+ df.y
df
```

```
          x         y
0      0.00    0.0000
1      0.01    0.0001
2      0.02    0.0004
3      0.03    0.0009
4      0.04    0.0016
...     ...       ...
996    9.96   99.2016
997    9.97   99.4009
998    9.98   99.6004
999    9.99   99.8001
1000  10.00  100.0000

[1001 rows x 2 columns]
```

```
df["soucin"] = df.x * df.y
df
```

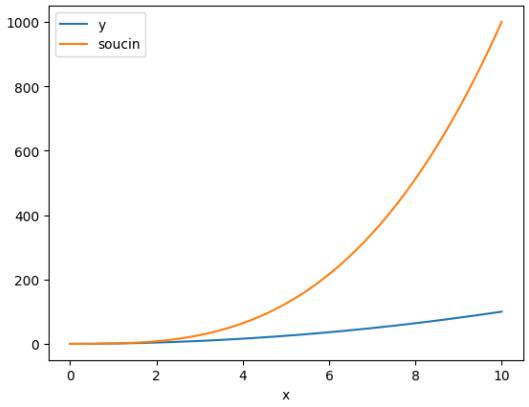
```
          x         y         soucin
0      0.00    0.0000       0.000000
1      0.01    0.0001       0.000001
2      0.02    0.0004       0.000008
3      0.03    0.0009       0.000027
4      0.04    0.0016       0.000064
...     ...       ...            ...
996    9.96   99.2016     988.047936
997    9.97   99.4009     991.026973
998    9.98   99.6004     994.011992
999    9.99   99.8001     997.002999
1000  10.00  100.0000    1000.000000

[1001 rows x 3 columns]
```

Tabulky je možno použít k operacím se sloupci podobně jako to znáte z Excelu, ale přehledněji.

> *Tabulky využijeme k ukládání stejně dlouhých datových řad. Výhodou je, že tabulky mají mnoho nástrojů na kreslení, manipulaci se sloupci a podobně.*

```
df.plot(x="x")
```

```
<Axes: xlabel='x'>
```




```
x = np.linspace(0,2*np.pi,100)
df = pd.DataFrame()
df["x"]=x
df["sin"] = np.sin(df.x)
df["cos"] = np.cos(df.x)
df["soucet mocnin"] = df.sin**2 + df.
↪cos**2
df
```

```
           x           sin       cos␣
↪ soucet mocnin
0   0.000000  0.000000e+00  1.000000␣
↪           1.0
1   0.063467  6.342392e-02  0.997987␣
↪           1.0
2   0.126933  1.265925e-01  0.991955␣
↪           1.0
```

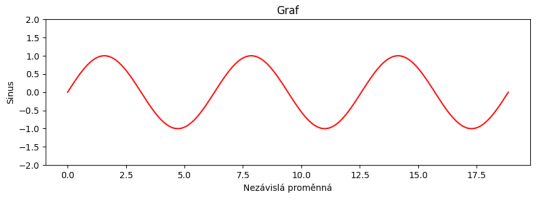
(continues on next page)

(pokračujte na předchozí stránce)

```
3   0.190400  1.892512e-01  0.981929␣
↪           1.0
4   0.253866  2.511480e-01  0.967949␣
↪           1.0
..       ...           ...       ...␣
↪           ...
95  6.029319 -2.511480e-01  0.967949␣
↪           1.0
96  6.092786 -1.892512e-01  0.981929␣
↪           1.0
97  6.156252 -1.265925e-01  0.991955␣
↪           1.0
98  6.219719 -6.342392e-02  0.997987␣
↪           1.0
99  6.283185 -2.449294e-16  1.000000␣
↪           1.0

[100 rows x 4 columns]
```

14.9 Obrázky

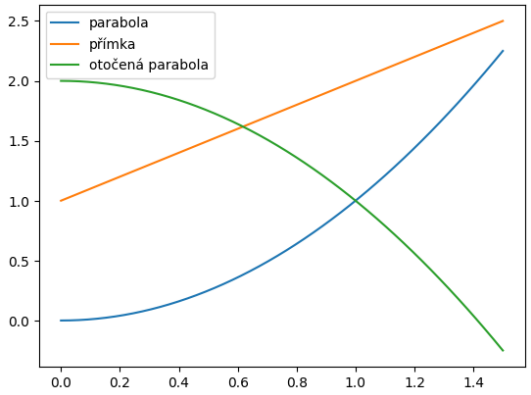
```
# vypocet dat
x = np.linspace(0,6*np.pi,1000) # data␣
↪na vstupu, definicni obor funkce
y = np.sin(x) # funkcni hodnoty

# vykresleni dat do obrazku
fig,ax = plt.subplots(figsize=(10,3))
plt.plot(x, y, color='red')

# kosmeticke upravy
ax.set(
    title='Graf',
    ylabel='Sinus',
    xlabel='Nezávislá proměnná',
    ylim=(-2,2)
);
```



14.10 Vykreslení dvourozměného pole

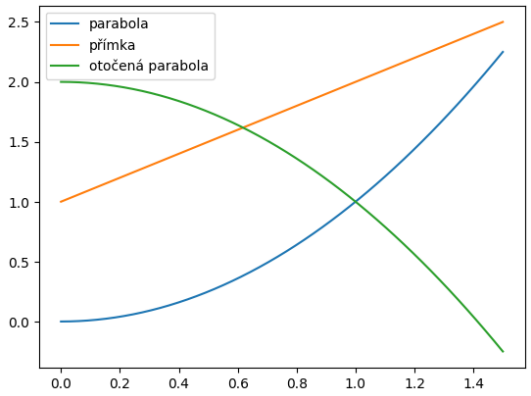
Pokud druhý argument není pole čísel, ale pole složené z polí, kreslí se příslušný počet křivek.

```
x = np.linspace(0,1.5)
seznam_funkci = [ [i**2,i+1,2-i**2] for␣
 ↪i in x ]
popisek = ["parabola","přímka",
 ↪"otočená parabola"]
plt.plot(x,seznam_funkci, label=popisek)
plt.legend();
```



Body do každé křivky se berou ze sloupců. Pokud tedy vytvoříme pole ze seznamu křivek, musíme jej transponovat, tj. řádky přepsat do sloupců. Aby se dalo pole transponovat, je nutné jej mít jako `np.array`.

```
x = np.linspace(0,1.5)
seznam_funkci = np.array([x**2,x+1,2-
 ↪x**2])
popisek = ["parabola", "přímka",
 ↪"otočená parabola"]
plt.plot(x,seznam_funkci.T,␣
 ↪label=popisek)
plt.legend();
```



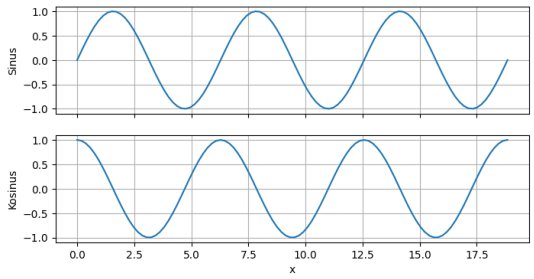
Uvedený postup využijeme u modelů, kde řešením dostaneme časový průběh více neznámých, například při modelování interakce mezi dvěma a více populacemi. Nemusíme kreslit každou křivku samostatně, ale nakreslíme je jedním příkazem.

14.11 Dva obrázky pod sebou, data z tabulky

Někdy chceme nakreslit do jednoho obrázku dvě funkce se společným definičním oborem, ale značně rozdílnými funkčními hodnotami. Řešením je buď nakreslit obrázky pod sebe a se sdílenou vodorovnou osou (viz níže), nebo nakreslit do jednoho obrázku obě funkce, ale každou s jiným měřítkem na svislé ose, tedy použít v jednom obrázku dvě svislé osy (viz Google a hesla `matplotlib` a `twinx`).

```
# tabulka funkcnich hodnot pro sinus a␣
 ↪kosinus
df = pd.DataFrame()
df["x"] = np.linspace(0,6*np.pi,100)
df["sin"] = np.sin(df.x)
df["cos"] = np.cos(df.x)

# Tisk začátku tabulky pro kontrolu
print(df.head())

# vykresleni dat do obrazku
fig,ax = plt.subplots(2,1,figsize=(8,4),
 ↪sharex=True) # zalozeni obrazku se␣
 ↪dvema soustavami souradnic pod sebou
df.plot(x="x",y="sin",ax=ax[0],␣
 ↪legend=False)# prvni graf
df.plot(x="x",y="cos",ax=ax[1],␣
 ↪legend=False)# druhy graf

# dekorace grafu
ax[0].grid() # vykresleni mrizky
ax[1].grid() # vykresleni mrizky
ax[0].set(ylabel="Sinus")
ax[1].set(ylabel="Kosinus");
```

```
          x       sin       cos
0  0.000000  0.000000  1.000000
1  0.190400  0.189251  0.981929
2  0.380799  0.371662  0.928368
3  0.571199  0.540641  0.841254
4  0.761598  0.690079  0.723734
```



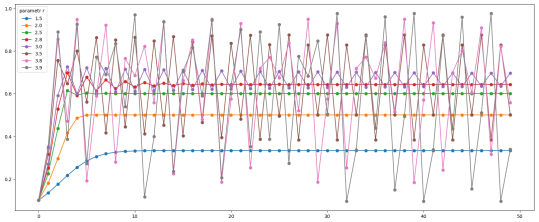
14.12 Růst populace v prostředí s omezenou nosnou kapacitou

Kód simuluje pro různé hodnoty parametru r chování modelu populace, vyvíjející se v prostředí s omezenou nosnou kapacitou podle vztahu

$$x_{k+1} = rx_k(1 - x_k).$$

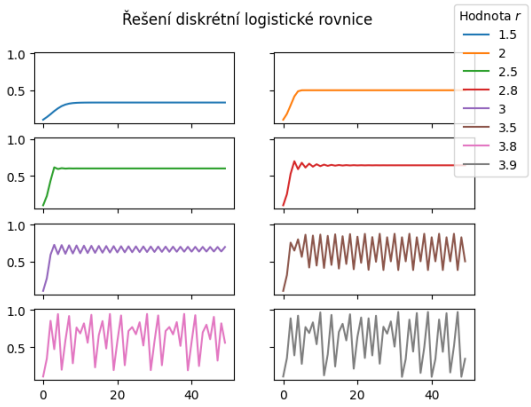
O matematických souvislostech se budeme bavit během semestru. Tento příklad ukazuje, že v rekurentních vzorcích se může objevit chaos. Pěkný model růstu populace pro malé hodnoty parametru je pro velké hodnoty nahrazen cykly přeskakujícími mezi několika hodnotami nebo dokonce chaosem. Viz Logistic map na Wikipedii.

```
N = 50
df = pd.DataFrame()
seznam_r = [1.5,2, 2.5, 2.8, 3, 3.5, 3.
↪8,3.9]

for r in seznam_r:
    x = np.zeros(N) # vytvoření seznamu␣
↪potřebné délky
    x[0]=0.1
    for i in range(N-1):
      x[i+1] = r*x[i]*(1-x[i])
    df[r] = x

fig,ax = plt.subplots(figsize=(20,8))
df.plot(ax=ax,style="o-")
plt.legend(title=r"parametr $r$");
```



Obrázky můžeme nakreslit také přehledněji do mřížky se sdílenými hodnotami na osách.

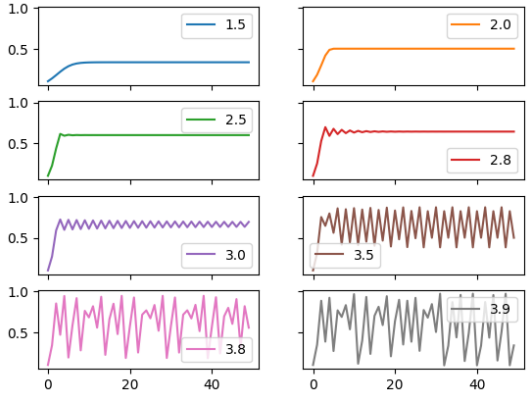
```
fig,axs = plt.subplots(int(len(seznam_
↪r)/2),2,sharex=True, sharey=True)
ax = axs.flatten()
for i,r in enumerate(seznam_r):
    ax[i].plot(df[r],label=r, color="C
↪"+str(i))
fig.legend(title=r"Hodnota $r$")
fig.suptitle("Řešení diskrétní␣
↪logistické rovnice");
```



14.13 Každá věc se dá udělat více způsoby ...

Do mřížky umí grafy uspořádat i přímo příkaz pro kreslení proměnné obsahující tabulku.

```
df.plot(
    subplots=True,
    layout=(int(len(seznam_r)/2),2),
    sharex=True,
    sharey=True);
```



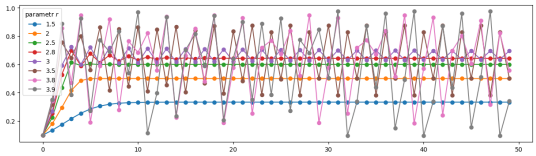
Je také možné počítat pro všechny hodnoty parametru r současně. Potom stačí jeden cyklus. Nemusí se dělat cyklus pro každou hodnotu parametru samostatně.

```
N = 50
df = pd.DataFrame()
seznam_r = [1.5,2, 2.5, 2.8, 3, 3.5, 3.
↪8,3.9]

x = np.zeros([N, len(seznam_r)]) #␣
↪vytvoření seznamu potřebné délky
x [0,:] = 0.1
for i in range(N-1):
    x[i+1,:] = seznam_r*x[i,:]*(1-x[i,
↪:])

fig,ax = plt.subplots(figsize=(15,4))
```

(continues on next page)

(pokračujte na předchozí stránce)

```
plt.plot(x,"o-")
plt.legend(seznam_r,title=r"parametr $r$
 ↪");
```



KAPITOLA 15

Derivace a modely založené na derivaci

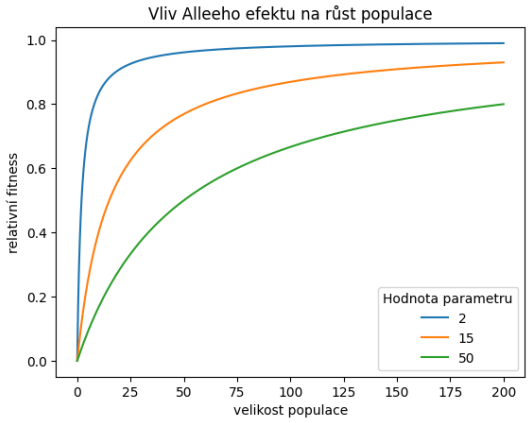
15.1 Úvod

Zopakujeme si vytváření tabulky a kreslení dat z tabulky z minulého cvičení. Poté si ukážeme, jak je možné definovat novou funkci.

15.1.1 Obrázek se třemi funkcemi

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

x = np.linspace(0,200,1000)
seznam_parametru = [2,15,50]
df = pd.DataFrame()
df["x"] = x
for parametr in seznam_parametru:
    df[parametr] = x/(parametr+x)

ax = df.plot(x="x")
ax.set(
    title="Vliv Alleeho efektu na růst␣
 ↪populace",
    xlabel="velikost populace",
    ylabel="relativní fitness")
plt.legend(title = "Hodnota parametru")
```

```
<matplotlib.legend.Legend at␣
 ↪0x7f31b3bf18d0>
```



15.1.2 Definice vlastní funkce

Delší úseky kódu zpravidla kumulujeme do funkcí. Tím se výsledný skript zjednoduší a je lépe srozumitelný.

Příklad definice funkce s povinnými a nepovinnými parametry

```
def mocnina(zaklad,n=2):
    vysledek = zaklad**n
    return vysledek
```

Ukázky volání. V prvním případě se použije přednastavená hodnota exponentu, ve druhém případě se umocňuje na čtvrtou.

```
mocnina(5)
```

```
25
```

```
mocnina(5,n=4)
```

```
625
```

Následující volání je zapoznámkováno, protože vyvolá chybu. Nebyl zadán základ a ten je povinným parametrem. Můžete zrušit zapoznámkování a příkaz si zkusit.

```
#mocnina()
```

Pokud zadáváme všechny volitelné parametry, není nutné psát jejich jméno. Níže je 6^7 (šest na sedmou).

```
mocnina(6,7)
```

```
279936
```

15.2 Numerická simulace v rovnici ochlazování

Studujme úlohu ochlazování kávy o teplotě T z počáteční teploty T_0 v prostředí s teplotou T_∞ a koeficientem k charakterizujícím intenzitu tepelné výměny podle Newtonova zákona tepelné výměny. Matematický model děje má tvar ve tvaru

$$\frac{\mathrm{d}T}{\mathrm{d}t} = -k\,(T - T_\infty), \quad T(0) = T_0.$$

Pokusíme se odhadnout po malých krůčcích chování řešení. Uvnitř každého krůčku budeme rychlost považovat za konstantní a změnu teploty určíme jako součin rychlosti a času (délky časového intervalu).

15.2.1 Jednoduchá numerická simulace

Vyřešíme rovnici pro počáteční teplotu $T_0 = 100°\mathrm{C}$, okolní teplotu $T_\infty = 20°\mathrm{C}$, koeficient přímé úměrnosti $k = 0.1\,\mathrm{min}^{-1}$ a časový krok $\Delta t = 2\,\mathrm{min}$.

Data budeme sestavovat do tabulky s časem a teplotou. První řádek je dán počáteční teplotou, časový sloupec je dán časovým krokem a musíme dopočítat teplotu.

t	T
0	100
2	
4	
6	
8	
…	

- Výpočet teploty v čase $2\,\mathrm{min}$
  - Rozdíl teplot je

    $$\Delta T = 100°\mathrm{C} - 20°\mathrm{C} = 80°\mathrm{C}.$$

  - Rychlost ochlazování je

    $$k(T - T_\infty) = 0.1\,\mathrm{min}^{-1} \times 80°\mathrm{C} = 8°\mathrm{C}/\mathrm{min}.$$

  - Změna teploty je

    $$\Delta T = -k\,(T - T_\infty)\Delta t = -8°\mathrm{C}/\mathrm{min} \times 2\,\mathrm{min} = -16°\mathrm{C}.$$

  - Teplota v čase $t = 2\,\mathrm{min}$ je součtem teploty v předešlém čase a změny teploty, tj.

    $$T(2) = T(0) + \Delta T = 100°\mathrm{C} - 16°\mathrm{C} = 84°\mathrm{C}.$$

    Nová tabulka je následující

t	T
0	100
2	84
4	
6	
8	
…	

- Výpočet teploty v čase $4\,\mathrm{min}$
  - Rozdíl teplot je

    $$\Delta T = 84°\mathrm{C} - 20°\mathrm{C} = 64°\mathrm{C}.$$

  - Rychlost ochlazování je

    $$k(T - T_\infty) = 0.1\,\mathrm{min}^{-1} \times 64°\mathrm{C} = 6.4°\mathrm{C}/\mathrm{min}.$$

  - Změna teploty je

    $$\Delta T = -k\,(T - T_\infty)\Delta t = -6.4°\mathrm{C}/\mathrm{min} \times 12\,\mathrm{min} = -12.8°\mathrm{C}$$

  - Teplota v čase $t = 4\mathrm{min}$ je součtem teploty v předešlém čase a změny teploty, tj.

    $$T(4) = T(2) + \Delta T = 84°\mathrm{C} - 12.8°\mathrm{C} = 71.2°\mathrm{C}.$$

    Nová tabulka je následující

t	T
0	100
2	84
4	71.2
6	
8	
…	

- Analogicky pokračujeme a dopočítáváme teplotu v dalších časech.

15.2.2 Numerická simulace po krocích konečné délky a srovnání s přesným řešením

```
# Nastavení parametrů
tmin = 0
tmax = 20
N = 11
k = 0.1
T0 = 100
T_inf = 20

# Pomocné proměnné
t = np.linspace(tmin,tmax,N) # časová␣
↪osa pro simulaci
dt = t[1]-t[0]

# Počáteční nastavení
T = np.zeros(N)  # pole pro ukládání␣
↪teplot
T[0] = T0  # počáteční teplota

# Simulace po časových krocích
for i in range(1,N):
    rozdil_teplot = T[i-1] - T_inf
    rychlost_ochlazovani = k*rozdil_
↪teplot
    zmena_teploty = - dt*rychlost_
↪ochlazovani
    T[i] = T[i-1] + zmena_teploty

# Uložení do tabulky
df = pd.DataFrame()
df["t"] = t
df["T"] = T
df.head()
```

```
     t        T
0  0.0  100.000
1  2.0   84.000
2  4.0   71.200
```

(continues on next page)

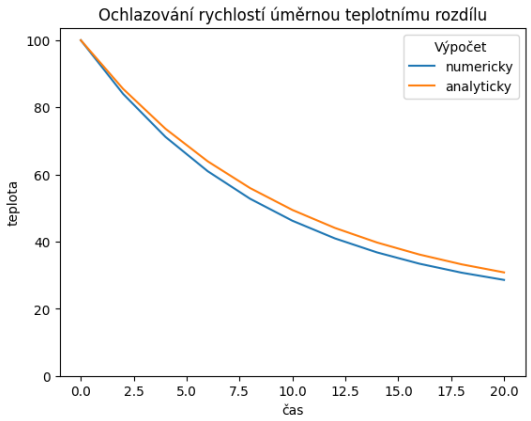
(pokračujte na předchozí stránce)

```
3  6.0   60.960
4  8.0   52.768
```

V tomto případě máme k dispozici i analytické řešení. To má tvar

$$T(t) = T_\infty + (T_0 - T_\infty)e^{-kt}.$$

```
# Analytické přesné řešení
df["analyticky"] = T_inf + (T0-T_
↪inf)*np.exp(-k*t)

# Vykreslení tabulky
df.plot(x="t")
plt.gca().set(
    xlabel="čas",
    ylabel="teplota",
    title="Ochlazování rychlostí␣
↪úměrnou teplotnímu rozdílu",
    ylim=(0,None)
)
plt.legend(['numericky','analyticky'],
↪title="Výpočet");
```



Abychom mohli pohodlně řešit modely pro různé parametry, můžeme příkazy z buňky, kde se v cyklu `for` počítají teploty pro jednotlivé okamžiky, seskupit do funkce a poté volat jedním příkazem. Modifikace spočívá v následujícím.

- Definice hlavičky funkce a popis činnosti funkce. Hlavička obsahuje volitelné parametry s přednastavenými hodnotami.
- Odsazení bloku s tělem funkce (označit blok a všechny řádky posunout naráz tabelátorem).
- Klíčové slovo `return` definující výstup.

```
def numericke_reseni(
    tmin = 0,
    tmax = 10,
    N = 10,
    k = 0.5,
```

(continues on next page)

(pokračujte na předchozí stránce)

```
    T0 = 100,
    T_inf = 20,
):
    """
    Naivní simulace Newtonova modelu␣
↪ochlazování. Derivace je nahrazena
    dopřednou diferencí.

    Na vstupu funkce jsou volitelné␣
↪parametry nastavující časový interval
    pro simulaci, dělení intervalu␣
↪udávající jemnost skoků, koeficient
    úměrnosti, počáteční teplota a␣
↪koncová teplota.

    Výstupem je tabulka (pandas.
↪DataFrame) se sloupci t a T pro čas a␣
↪teplotu.
    """
    t = np.linspace(tmin,tmax,N) #␣
↪časová osa pro simulaci
    dt = t[1]-t[0]

    # Počáteční nastavení
    T = np.zeros(N)  # pole pro␣
↪ukládání teplot
    T[0] = T0        # počáteční teplota

    # Simulace po časových krocích
    for i in range(1,N):
        rozdil_teplot = T[i-1] - T_inf
        rychlost_ochlazovani = k*rozdil_
↪teplot
        zmena_teploty = - dt*rychlost_
↪ochlazovani
        T[i] = T[i-1] + zmena_teploty

    # Uložení do tabulky
    df = pd.DataFrame()
    df["t"] = t
    df["T"] = T

    return df
```

Ukázka volání, všechny parametry necháme na defaultních hodnotách.

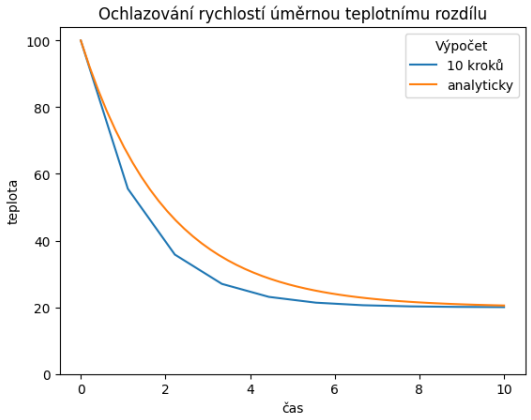
```
numericke_reseni()
```

	t	T
0	0.000000	100.000000
1	1.111111	55.555556
2	2.222222	35.802469
3	3.333333	27.023320
4	4.444444	23.121475
5	5.555556	21.387322
6	6.666667	20.616588
7	7.777778	20.274039
8	8.888889	20.121795
9	10.000000	20.054131

Úkol Experimentujte s počtem bodů N a tím i s délkou kroku. Sledujte hladkost numerického řešení a jeho odchylku od přesného analytického řešení.

V praxi musíme mít krok dostatečně malý, aby řešení bylo hladké a přesné, ale ne moc malý, aby nás to nestálo hodně paměti, výpočetního výkonu, času a aby nehrozilo, že se při mnoha výpočtech akumulují zaokrouhlovací chyby. Zpravidla nám toto obstarají procedury pro řešení automaticky.

```
# Měňte délku kroku a sledujte výstup␣
↪(méně zdatní), nebo vykreslete do␣
↪jednoho
# obrázku průběh pro více kroků␣
↪(zdatnější - stačí opakovat tři řádky␣
↪s
# jinou hodnotou N, s~výpočtem řešení a␣
↪vykreslením)

# Nastavení parametrů
k = 0.5
tmin = 0
tmax = 10
T0 = 100
T_inf = 20


# Numerické řešení
N = 10
r = numericke_reseni(N=N, k=k,␣
↪tmin=tmin, tmax=tmax, T0=T0, T_inf=T_
↪inf)  # vyřešení modelu
plt.plot(r["t"],r["T"], label=f"{N}␣
↪kroků")      # vykreslení řešení

# Analytické řešení
t = np.linspace(tmin,tmax)
plt.plot(t,T_inf + (T0-T_inf)*np.exp(-
↪k*t), label="analyticky")

# Úpravy grafu
ax = plt.gca()                # uložení␣
↪souřadné soustavy do vlastní proměnné␣
↪pro kosmetiku
ax.set(                       #␣
↪modifikace souřadné soustavy,␣
↪kosmetické úpravy
    xlabel="čas",
    ylabel="teplota",
    title="Ochlazování rychlostí␣
↪úměrnou teplotnímu rozdílu",
    ylim=(0,None)
)
plt.legend(title="Výpočet");
```



15.3 Simulace pro více počátečních podmínek

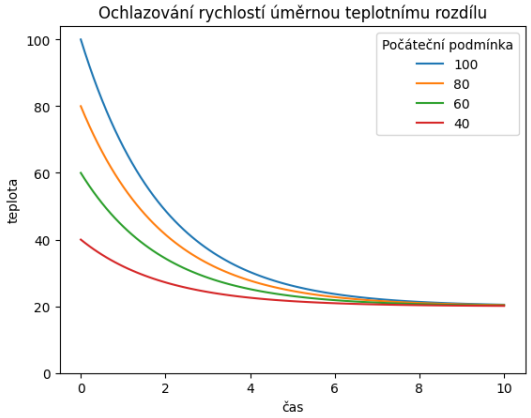
Vyjdeme z předchozí simulace, ale novou simulaci spustíme v cyklu. Jeden průběh cyklu pro každou počáteční podmínku. Výstup budeme ukládat do nové tabulky a tu potom vykreslíme.

```
# Tip: vygenerovaný obrázek nepotřebuje␣
↪legendu ani změnu barev. Zkuste␣
↪legendu
# vypnout a zakreslit všechny křivky␣
↪stejnou barvou. Například barvou C0
# (základní barva pro první křivku␣
↪obrázku).
# https://pandas.pydata.org/docs/
↪reference/api/pandas.DataFrame.plot.
↪html
# https://matplotlib.org/stable/api/_as_
↪gen/matplotlib.pyplot.plot.html

# Příprava proměnných
T0_seznam = [100, 80, 60, 40]
df2 = pd.DataFrame()

# Výpočet
for T0 in T0_seznam:
    reseni = numericke_reseni(T0=T0,␣
↪N=100)
    df2[T0] = reseni["T"]
df2["t"] = reseni["t"]

# Vykreslení tabulky
df2.plot(x="t")

# Kosmetika
ax = plt.gca()
ax.set(
    xlabel="čas",
    ylabel="teplota",
    title="Ochlazování rychlostí␣
↪úměrnou teplotnímu rozdílu",
    ylim=(0,None)
```

(continues on next page)

(pokračujte na předchozí stránce)

```
)
plt.legend(title="Počáteční podmínka");
↪# nadpis pro legendu
```



15.3.1 Úkol: simulace pro měnící se koeficient úměrnosti

Udělejte simulaci s jednou počáteční podmínkou a měnící se hodnotu koeficientu k. Nakopírujte si sem předchozí buňku a modifikujte tak, aby byla počáteční podmínka pořád stejná a měnil se koeficient k. Musíte sestavit seznam hodnot pro proměnnou k, změnit cyklování a změnit jméno sloupce v tabulce tak, aby zachycovalo hodnotu k.

```
# Jako výchozí sem nakopírujte obsah␣
↪buňky, která kreslí řešení pro různé
# počáteční podmínky. Tento kód␣
↪modifikujte tak, aby počáteční␣
↪podmínka byla
# stále stejná, ale měnila se hodnota k.
```

15.4 Simulace pro model se zpožděním

Uvedené simulace jsou naivní metodou řešení úlohy modelující pomocí derivací nějaký proces. Pro řešení modelů s derivacemi existují mnohem vyspělejší metody, které si představíme příště. V praxi se zpravidla spoléháme na tyto metody, které jsou naprogramované profesionály.

Dovednost umět si implementovat jednoduchý řešič může sloužit pro orientační první nástřel výpočtu, nebo při potřebě řešení nějaké speciální úlohy. Jedním takovým speciálním postupem může být snaha započítat do rovnice ochlazování zpoždění. V takovém případě rychlost změny není dána aktuálními hodnotami, ale hodnotami ve minulosti. To by mohlo odpovídat regulaci teploty, kdy

se změna v nastavení neprojeví okamžitě. Všichni takovou situaci známe při nastavování teploty ve sprše.

Model ochlazování se zpožděním θ má tvar

$$\frac{\mathrm{d}T}{\mathrm{d}t} = -k\,(T(t-\theta) - T_\infty), \quad T(0) = T_0$$

a při numerickém řešení je jediný rozdíl v tom, že rychlost nestanovujeme podle teploty na předešlém řádku tabulky, ale používáme některý z předešlých řádků, v závislosti na velikosti zpoždění.

```
def numericke_reseni_se_zpozdenim(  # !!
↪! jiné jméno funkce
    tmin = 0,
    tmax = 10,
    N = 10,
    k = 0.5,
    T0 = 100,
    T_inf = 20,
    zpozdeni = 0      # !!! další␣
↪parametr
):
    """
    Stejná funkce jako numericke_reseni␣
↪s~dodatečným parametrem pro zpoždění.
    Rozdíly jsou vyznačeny třemi␣
↪vykřičníky.
    """
    t = np.linspace(tmin,tmax,N) #␣
↪časová osa pro simulaci
    dt = t[1]-t[0]
    zpozdeni_index = int(zpozdeni/dt) #␣
↪!!! převod času na počet hodnot zpět

    # Počáteční nastavení
    T = np.zeros(N)  # pole pro␣
↪ukládání teplot
    T[0] = T0        # počáteční teplota

    # Simulace po časových krocích
    for i in range(1,N):
        # Teplota nemusí být teplota␣
↪v~předchozím bodě, ale teplota dříve␣
↪v
        # historii. Musíme ošetřit, aby␣
↪se index nedostal do záporných hodnot.
        # (To nastane pro několik␣
↪prvních hodnot.)
        predchozi_index = max(i-1-
↪zpozdeni_index ,0) # !!!
        rozdil_teplot = T[predchozi_
↪index] - T_inf   # !!!
        rychlost_ochlazovani = k*rozdil_
↪teplot
        zmena_teploty = - dt*rychlost_
↪ochlazovani
        T[i] = T[i-1] + zmena_teploty

    # Uložení do tabulky
    df = pd.DataFrame()
    df["t"] = t
```

(continues on next page)

(pokračujte na předchozí stránce)

```
    df["T"] = T

    return df
```

```
seznam_zpozdeni = [0,0.1,0.5,1,2]
df3 = pd.DataFrame()

for zpozdeni in seznam_zpozdeni:
    reseni = numericke_reseni_se_
↪zpozdenim(zpozdeni=zpozdeni,N=1000)
    df3[zpozdeni] = reseni["T"]
df3["t"] = reseni["t"]

df3
```

```
            0.0          0.1          ␣
↪0.5          1.0          2.0          ␣
↪ t
0    100.000000  100.000000  100.
↪000000  100.000000  100.000000   0.
↪00000
1     99.599600   99.599600   99.
↪599600   99.599600   99.599600   0.
↪01001
2     99.201203   99.199199   99.
↪199199   99.199199   99.199199   0.
↪02002
3     98.804801   98.798799   98.
↪798799   98.798799   98.798799   0.
↪03003
4     98.410382   98.398398   98.
↪398398   98.398398   98.398398   0.
↪04004
..          ...          ...          .
↪..          ...          ...         ..
↪.
995   20.543102   20.426601   20.
↪072807   20.021464   32.857099   9.
↪95996
996   20.540384   20.424362   20.
↪072290   20.021116   32.775063   9.
↪96997
997   20.537679   20.422135   20.
↪071778   20.020770   32.692684   9.
↪97998
998   20.534988   20.419920   20.
↪071268   20.020428   32.609967   9.
↪98999
999   20.532311   20.417716   20.
↪070763   20.020088   32.526917  10.
↪00000

[1000 rows x 6 columns]
```

```
df3.plot(x="t")
plt.legend(title="Zpoždění")
plt.title("Rovnice ochlazování se␣
↪zpožděním")
```

```
Text(0.5, 1.0, 'Rovnice ochlazování␣
↪se zpožděním')
```

Úkol Předchozí buňky vytvoří tabulku s daty a poté vykreslí graf. To dá přehled o tom, že opravdu vzniká tabulka tak jak má. Pokud chceme sledovat, jaký mají změny dalších parametrů vliv na graf, je lepší obě buňky spojit, aby se generování tabulky i grafu spouštělo současně. Vyzkoušete to. Například tak, že se nastavíte na horní buňku a stisknete Shift+M. (Pozor na to, bez shiftu byste buňku přepnuli na textovou.)

KAPITOLA 16

Diferenciální rovnice 1

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy.integrate import solve_ivp
```

- Ve cvičení začínáme částí A, řešení modelu dynamické rovnováhy druhů z přednášky.
- Kdo je schopen postupovat podle textu a řešit úkoly samostatně, může pracovat dopředu a pokračovat částí B, kde jsou na modelu ochlazování kávy možnosti příkazu pro řešení diferenciálních rovnic.
- Část C se vrací k modelu dynamické rovnováhy, ukazuje příklad, jak je možné nakreslit křivky řídící rychlosti kolonizace a vymírání a jak je možné prozkoumat chování modelu při různých počátečních podmínkách a různých hodnotách parametru. Zájemci by měli být schopni si tyto pasáže projít sami.

16.1 Část A: Řešení diferenciální rovnice (modifikace existujícího kódu)

Pokud jste rychlejší, řešte samostatně a pokračujte samostatně k části B s detailnějším popisem příkazu `solve_ivp`.

16.1.1 Řešení rovnice a vizualizace řešení.

Následující kód vyřeší rovnici

$$\frac{\mathrm{d}N}{\mathrm{d}t} = \frac{b}{D(N+\beta)} - a\frac{N^k}{S}$$

dynamické rovnováhy na ostrově s nulovou počáteční podmínkou.

Úkoly:

- Vyzkoušejte si. Zkuste i zadání více počátečních podmínek, například řádek `pocatecni_podminka = [0]` vyměňte za `pocatecni_podminka = [0,5,10,20]`. Tím získáte řešení pro několik počátečních podmínek současně.
- Vykopírujte text do nové buňky a opravte tak, aby zobrazoval řešení počáteční úlohy

  $$\frac{\mathrm{d}T}{\mathrm{d}t} = -0.1(T-20), \quad T(0) = 100$$

  s ochlazováním kávy z minulého cvičení. I zde zkuste více počátečních podmínek.

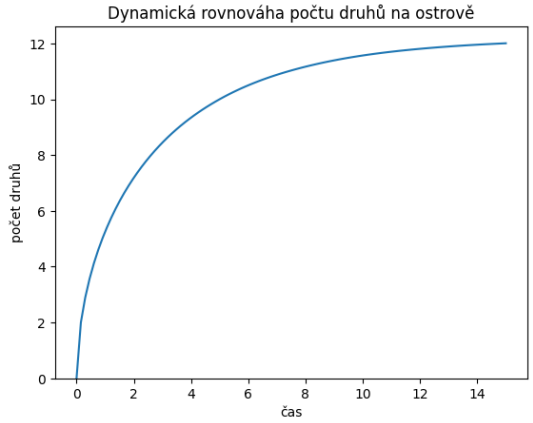
```
### Příprava funkcí a parametrů
pocatecni_podminka = [0]  # počáteční␣
↪podmínka nebo podmínky
meze = [0,15]  # interval, na kterém␣
↪hledáme řešení
n = 100 # počet dělících bodů

def model(t, N, a=1, b=8, beta=0.2, D=0.
↪5, k=1.3, S=20):
    """
    Funkce z~pravé strany modelu␣
↪dynamické rovnováhy počtu druhů na␣
↪ostrovech,
    podle McArthura a Wilsona.
```

(continues on next page)

(pokračujte na předchozí stránce)

```
    Vstup:
    -----
    Povinnými parametry jsou čas a␣
 ↪počet druhů, volitelnými vzdálenost
    D od pevniny, rozloha ostrova S,␣
 ↪další parametry modelu a konstanty
    úměrnosti. Přednastavené hodnoty␣
 ↪jsou pouze ilustrační, závisí na volbě
    jednotek a konkrétním použití.

    Výstup:
    ------
    Hodnota funkce.
    """
    kolonizace = b/(D*(N+beta))
    vymirani = a*N**k/S
    return  kolonizace - vymirani

### Řešení modelu
t=np.linspace(*meze, n)  # definiční␣
 ↪obor, v~těchto bodech budeme hledat␣
 ↪řešení
reseni = solve_ivp(
                  model,
                  meze,
                  pocatecni_podminka,
                  t_eval=t
                  )

### Vizualizace řešení
fig,ax = plt.subplots(1)
ax.plot(t,reseni.y.T)
ax.set(
    ylim = (0,None),
    title = "Dynamická rovnováha počtu␣
 ↪druhů na ostrově",
    xlabel="čas",
    ylabel="počet druhů",
)
```

```
[(0.0, 12.609006758548855),
 Text(0.5, 1.0, 'Dynamická rovnováha␣
 ↪počtu druhů na ostrově'),
 Text(0.5, 0, 'čas'),
 Text(0, 0.5, 'počet druhů')]
```



16.1.2 Řešení pro sadu parametrů

Následující kód řeší vícekrát rovnici dynamické rovnováhy pro několik různých vzdálenosti ostrova od pevniny.

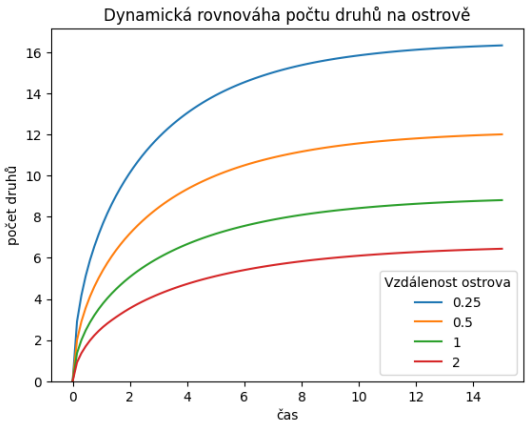
```
### Příprava funkcí a parametrů
pocatecni_podminka = [0]  # počáteční␣
 ↪podmínka
meze = [0,15]  # interval, na kterém␣
 ↪hledáme řešení
n = 100 # počet dělících bodů
parametry = [0.25,0.5,1,2] # seznam␣
 ↪parametrů


def model(t, N, a=1, b=8, beta=0.2, D=0.
 ↪5, k=1.3, S=20):
    """
    Funkce z~pravé strany modelu␣
 ↪dynamické rovnováhy počtu druhů na␣
 ↪ostrovech,
    podle McArthura a Wilsona.

    Vstup:
    -----
    Povinnými parametry jsou čas a␣
 ↪počet druhů, volitelnými vzdálenost
    D od pevniny, rozloha ostrova S,␣
 ↪další parametry modelu a konstanty
    úměrnosti. Přednastavené hodnoty␣
 ↪jsou pouze ilustrační, závisí na volbě
    jednotek a konkrétním použití.

    Výstup:
    ------
    Hodnota funkce.
    """
    kolonizace = b/(D*(N+beta))
    vymirani = a*N**k/S
    return  kolonizace - vymirani

### Řešení modelu
t=np.linspace(*meze, n)  # definiční␣
 ↪obor, v~těchto bodech budeme hledat␣
 ↪řešení
```

(continues on next page)

(pokračujte na předchozí stránce)

```
df = pd.DataFrame()       # tabulka pro␣
↪výstup
df["t"] = t               # sloupec␣
↪s~časem

for parametr in parametry:
    reseni = solve_ivp(
                          lambda t,
↪x:model(t,x,D=parametr),
                          meze,
                          pocatecni_
↪podminka,
                          t_eval=t
                          )
    df[parametr] = reseni.y.T # další␣
↪sloupec tabulky
    # lambda funkce viz https://www.
↪w3schools.com/python/python_lambda.asp
    # (dočasná nepojmenovaná funkce)

### Vizualizace řešení
ax = df.plot(x="t")
ax.set(
    ylim = (0,None),
    title = "Dynamická rovnováha počtu␣
↪druhů na ostrově",
    xlabel="čas",
    ylabel="počet druhů",
)
plt.legend(title="Vzdálenost ostrova")
```

```
<matplotlib.legend.Legend at␣
↪0x7f9003e6b0d0>
```



Úkoly

- Vyzkoušejte si kód.
- V nové buňce sledujte vliv rozlohy ostrova (vzdálenost je stejná) na druhovu skladbu. Vykopírujte si kód a proveďte příslušnou modifikaci.
- Před každou buňku s výpočty vložte textovou buňku popisující, co se ve výpočtu odehrává, co se snažíme ukázat. Musíte vložit buňku, změnit typ z Code na Markdown (vybrat buňku a stisknout M, nebo použít rozbalovací menu v toolbaru nebo menu Cell > Cell type > Markdown) a vepsat komentář.
- Pokud jsou veličiny S a D ve stále stejném poměru, je stejná i hodnota, ke které konverguje řešení (viz přednáška).
  - Znamená to, že se budou populace vyvíjet stejně na daném ostrově a na ostrově, který je dvakrát větší a dvakrát dále? V čem se bude situace lišit a v čem bude stejná?
  - Odhadněte odpověď a potvrďte si hypotézu tak, že budete modelovat plnou diferenciální rovnici pro obě uvažované situace. Můžete použít něco jako `lambda t,x:model(t, x,D=0.5*nasobek,S=20*nasobek)` a proměnnou `nasobek` nechat iterovat přes seznam `[1,2]` (případně delší seznam).

16.2 Část B: Řešení diferenciální rovnice (podrobnější pohled pod kapotu)

Příkaz `solve_ivp` dokáže vyřešit zadanou diferenciální rovnici pro několik počátečních podmínek. Použití je možné vidět na následujícím příkladě. Pro tři počáteční podmínky $y(0) = 2$, $y(0) = 4$ a $y(0) = 8$ řešíme rovnici

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -0.5y,$$

kterou je možno chápat jako rovnici ochlazování, nebo jako rovnici radioaktivního rozpadu. Výstupem je objekt, který v sobě obsahuje kromě vypočtených dat i další informace, například zda se podařilo rovnici vyřešit.

```
pocatecni_podminky = [2, 4, 8]
meze = [0,10]
def rovnice(t, y):
    return -0.5 * y
sol = solve_ivp(rovnice, meze,␣
↪pocatecni_podminky)
sol
```

```
 message: The solver successfully␣
↪reached the end of the integration␣
↪interval.
 success: True
  status: 0
       t: [ 0.000e+00  1.149e-01  1.
↪264e+00  3.061e+00  4.816e+00
            6.574e+00  8.333e+00  1.
↪000e+01]
       y: [[ 2.000e+00  1.888e+00 ..
↪.  3.107e-02  1.351e-02]
           [ 4.000e+00  3.777e+00 ..
↪.  6.214e-02  2.702e-02]
           [ 8.000e+00  7.553e+00 ..
↪.  1.243e-01  5.403e-02]]
     sol: None
t_events: None
y_events: None
    nfev: 44
    njev: 0
     nlu: 0
```

Následující příkazy vytisknou časy ve kterých bylo vypočteno řešení a hodnotu každého z řešení v daném čase.

```
print(sol.t)
print(sol.y)
```

```
[ 0.          0.11487653  1.26364188␣
↪ 3.06061781  4.81611105  6.57445806
  8.33328988 10.        ]
[[2.          1.88836035 1.06327177 0.
↪43319312 0.18017253 0.07483045
  0.03107158 0.01350781]
 [4.          3.7767207  2.12654355 0.
↪86638624 0.36034507 0.14966091
  0.06214316 0.02701561]
 [8.          7.5534414  4.25308709 1.
↪73277247 0.72069014 0.29932181
  0.12428631 0.05403123]]
```

Pro pohodnější práci je možné data zadat do tabulky, kde v prvním sloupci bude čas a v dalších sloupcích budou vypočtená řešení. Název sloupce bude dán počáteční podmínkou. Je vidět, že tímto číslem každé řešení i začíná.

```
df = pd.DataFrame()
df["t"] = sol.t
for i,j in zip(pocatecni_podminky, sol.
↪y):
    df[i] = j
df
```

```
          t         2         4       ␣
↪    8
0   0.000000  2.000000  4.000000  8.
↪000000
1   0.114877  1.888360  3.776721  7.
```

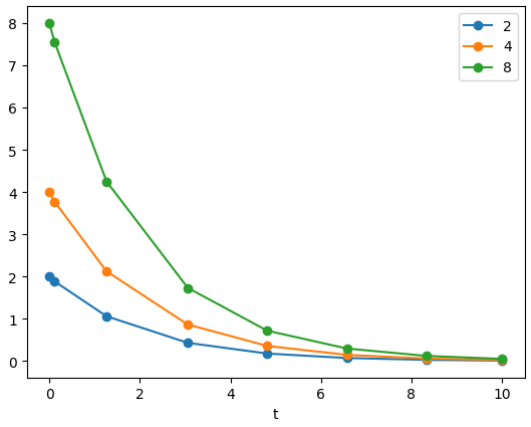
(continues on next page)

(pokračujte na předchozí stránce)

```
↪553441
2   1.263642  1.063272  2.126544  4.
↪253087
3   3.060618  0.433193  0.866386  1.
↪732772
4   4.816111  0.180173  0.360345  0.
↪720690
5   6.574458  0.074830  0.149661  0.
↪299322
6   8.333290  0.031072  0.062143  0.
↪124286
7  10.000000  0.013508  0.027016  0.
↪054031
```

Po vykreslení vidíme, že graf je z lomených čar. Pro výraznost jsme přidali i tečky v bodech zlomu. To proto, že řešení byla vypočtena a v několika málo bodech.

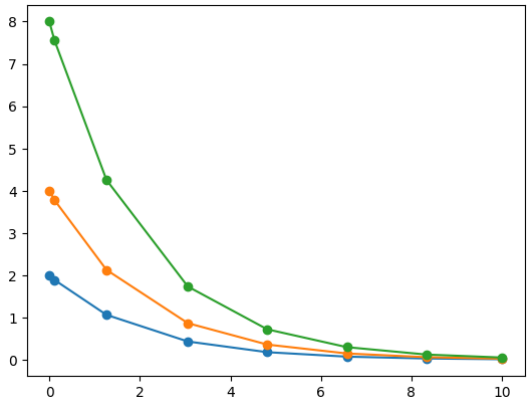
```
df.plot(x="t", style="o-");
```



16.2.1 Úkol

Vyzkoušejte si následující elegantnější příkazy. Pomocí nich nemusíme dělat cyklus přes jednotlivá řešení při sestavování tabulky a můžeme nakreslit všechna tři řešení jedním příkazem i bez sestavování tabulky. Snažte se zjistit, co dělá operátor `T`.

```
plt.plot(sol.t,sol.y.T,"o-");
```



```
df = pd.DataFrame(sol.y.T,
 ↪columns=pocatecni_podminky)
df["t"] = sol.t
df
```

```
          2         4         8      ␣
 ↪    t
0  2.000000  4.000000  8.000000   0.
 ↪000000
1  1.888360  3.776721  7.553441   0.
 ↪114877
2  1.063272  2.126544  4.253087   1.
 ↪263642
3  0.433193  0.866386  1.732772   3.
 ↪060618
4  0.180173  0.360345  0.720690   4.
 ↪816111
5  0.074830  0.149661  0.299322   6.
 ↪574458
6  0.031072  0.062143  0.124286   8.
 ↪333290
7  0.013508  0.027016  0.054031  10.
 ↪000000
```

```
pd.DataFrame(sol.y)
```

```
     0         1         2         3␣
 ↪         4         5         6      ␣
 ↪    7
0  2.0  1.888360  1.063272  0.433193␣
 ↪ 0.180173  0.074830  0.031072  0.
 ↪013508
1  4.0  3.776721  2.126544  0.866386␣
 ↪ 0.360345  0.149661  0.062143  0.
 ↪027016
2  8.0  7.553441  4.253087  1.732772␣
 ↪ 0.720690  0.299322  0.124286  0.
 ↪054031
```

```
pd.DataFrame(sol.y.T)
```

```
          0         1         2
0  2.000000  4.000000  8.000000
```

(continues on next page)

(pokračujte na předchozí stránce)

```
1  1.888360  3.776721  7.553441
2  1.063272  2.126544  4.253087
3  0.433193  0.866386  1.732772
4  0.180173  0.360345  0.720690
5  0.074830  0.149661  0.299322
6  0.031072  0.062143  0.124286
7  0.013508  0.027016  0.054031
```

16.2.2 Řešení hladšími křivkami

Řešení z minulých ukázek se skládalo z lomených čar. Nemělo spojitou derivaci, nebylo hladké. Pro hladší řešení je možné provést jednu nebo více z následujících vylepšení.

- Zvolit menší krok, aby se vygenerovalo více hodnot pro čas t,
- Nastavit proměnnou `t_eval` na dostatečně hustou posloupnost bodů. Toto jsme si vyzkoušeli v úvodní části.
- Nastavit hodnotu `dense_output` na True a řešení poté kreslit na husté množině bodů. Toto si vyzkoušíme níže.

Volba kroku také bývá základním testem korektnosti numerického řešení. Příliš velký krok může způsobit nepřesnosti v řešení, příliš malý krok je náročný na paměť i výpočetní výkon a také může vést k nepřesnostem. Obvyklým testem je porovnat řešení se zvoleným a s poloviční krokem. Pokud se shodují, je rozumné výsledek přijmout jako dobrou aproximaci přesného řešení.

```
sol = solve_ivp(
    rovnice,
    meze,
    pocatecni_podminky,
    dense_output=True,
    max_step=np.Inf, # defaultní␣
 ↪nastavení je krok libovolné délky
    )
sol
```

```
 message: The solver successfully␣
↪reached the end of the integration␣
↪interval.
  success: True
   status: 0
        t: [ 0.000e+00  1.149e-01  1.
↪264e+00  3.061e+00  4.816e+00
             6.574e+00  8.333e+00  1.
↪000e+01]
        y: [[ 2.000e+00  1.888e+00 ..
↪.  3.107e-02  1.351e-02]
            [ 4.000e+00  3.777e+00 ..
↪.  6.214e-02  2.702e-02]
            [ 8.000e+00  7.553e+00 ..
```

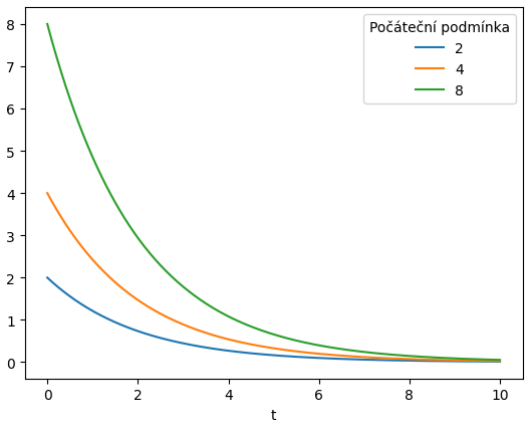
(continues on next page)

(pokračujte na předchozí stránce)

```
↪.  1.243e-01  5.403e-02]]
     sol: <scipy.integrate._ivp.
↪common.OdeSolution object at␣
↪0x7f900103f790>
t_events: None
y_events: None
    nfev: 44
    njev: 0
     nlu: 0
```

```
t = np.linspace(*meze,500)
df = pd.DataFrame(sol.sol(t).T,
 ↪columns=pocatecni_podminky)
df["t"] = t

print(df.shape)
print(df.head())

ax = df.plot(x="t")
ax.legend(title="Počáteční podmínka");
```

```
(500, 4)
          2         4         8      ␣
 ↪  t
0  2.000000  4.000000  8.000000  0.
 ↪00000
1  1.980060  3.960120  7.920240  0.
 ↪02004
2  1.960319  3.920638  7.841275  0.
 ↪04008
3  1.940774  3.881549  7.763098  0.
 ↪06012
4  1.921425  3.842850  7.685699  0.
 ↪08016
```

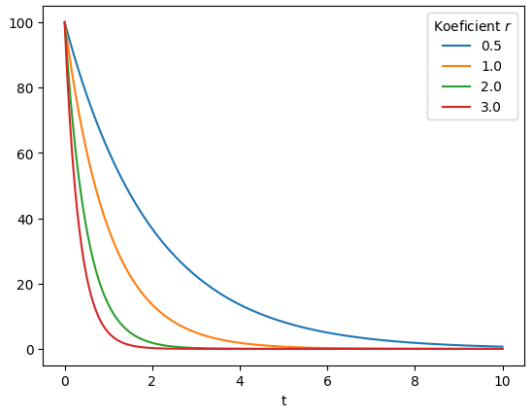


16.2.3 Řešení rovnice pro různé hodnoty parametru

Někdy potřebujeme řešit rovnici a sledovat, jak se chová řešení při změně parametru. Vyjdeme ze stejné počáteční podmínky a pro nastavené hodnoty parametru najdeme řešení.

```
def rovnice_r(t, y, r=1):
    return -r*y # prava strana rovnice␣
 ↪zavisi na parametru r

meze = [0,10]
hodnoty_r = [0.5,1,2,3]
pocatecni_podminka = [100] # Jedna␣
 ↪pocatecni podminka

t = np.linspace(*meze,500) # hvezdicka␣
 ↪rozbali pole na dve hodnoty pro dolni␣
 ↪a horni mez

reseni = [solve_ivp(
                    rovnice_r,
                    meze,
                    pocatecni_podminka,
                    dense_output=True,
                    args=[r]
                    )
            for r in hodnoty_r]
        # vygenerovani reseni pro ruzne␣
 ↪hodnoty parametru r

df = pd.DataFrame(
                    np.array([res.
 ↪sol(t)[0] for res in reseni]).T,
                    columns=hodnoty_r
                )
df["t"] = t

### Jiná alternativa uložení dat do␣
 ↪tabulky. Ukládáme postupně sloupce
### i s~jejich názvy do tabulky v~cyklu␣
 ↪for.
# df = pd.DataFrame() # tabulka pro␣
 ↪uložení řešení
# df["t"] = t          # sloupec s~časem
# for i,j in zip(hodnoty_r, reseni):
#     df[i] = j.sol(t)[0]  # sloupce␣
 ↪s~řešeními, v~záhlaví hodnota␣
 ↪parametru


print(df.shape) # tisk informaci␣
 ↪o~tabulce a graf
print(df.head())

ax = df.plot(x="t")
ax.legend(title=r"Koeficient $r$");
```

```
(500, 5)
          0.5          1.0          2.
 ↪0          3.0          t
0   100.000000   100.000000   100.
 ↪000000   100.000000   0.00000
1    99.002999    98.015939    96.
 ↪071240    94.165121   0.02004
2    98.015939    96.071242    92.
 ↪296834    88.670711   0.04008
3    97.038719    94.165130    88.
 ↪670717    83.496898   0.06012
4    96.071242    92.296836    85.
 ↪186977    78.622903   0.08016
```



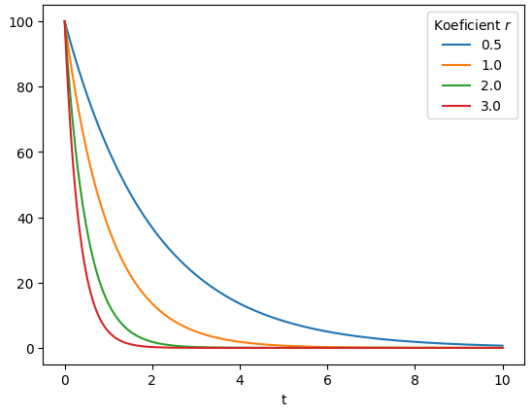
Někdy se může hodit si funkci definující pravou stranu nepojmenovávat a nepředávat jí parametry. Například, pokud je její použití jednorázové. Potom je možno použít metodiku práce s nepojmenovanými funkcemi, takzvané `lambda` funkce. Ta umožňuje zadat pravou stranu rovnice přímo v příkazu `solve_ivp`.

```
hodnoty_r = [0.5,1,2,3]
pocatecni_podminka = [100] # Jedna␣
 ↪pocatecni podminka

reseni = [solve_ivp(
                    lambda t,y: -r*y,
                    meze,
                    pocatecni_podminka,
                    dense_output=True,
                    )
            for r in hodnoty_r]
        # vygenerovani reseni pro ruzne␣
 ↪hodnoty parametru r

t = np.linspace(*meze,500) # hvezdicka␣
 ↪rozbali pole na dve hodnoty pro dolni␣
 ↪a horni mez
df = pd.DataFrame(
                    np.array(
                        [res.sol(t)[0]␣
 ↪for res in reseni]
                        ).T,
                    columns=hodnoty_r
                )
df["t"] = t
```

(continues on next page)

(pokračujte na předchozí stránce)

```
ax = df.plot(x="t")
ax.legend(title=r"Koeficient $r$");
```



16.3 Část C: Model rovnováhy počtu druhů na ostrovech

V této části již nebude nic nového, ale je to ukázka, jak získané znalosti využít při studiu modelů.

Budeme studovat Mc Arthurův a Wilsonův model vývoje ostrovního společenství ve tvaru

$$\frac{\mathrm{d}N}{\mathrm{d}t} = \frac{b}{D(N+\beta)} - a\frac{N^k}{S},$$

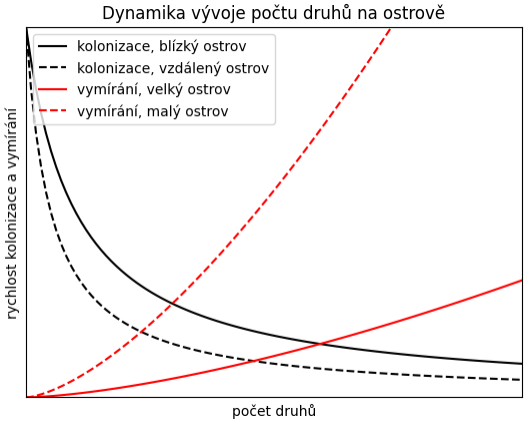
kde N je počet druhů na ostrově o velikosti S ve vzdálenosti D od pevniny. Všechny parametry a, b, β a k jsou kladné, parametr k splňuje $k > 1$.

První člen na pravé straně charakterizuje rychlost kolonizace, druhý člen rychlost vymírání.

- Nakreslíme si křivky kolonizace a vymírání.
  - Nejprve překreslíme obrázek, který se často vyskytuje v publikacích k problematice. Dvě křivky vymírání pro dvě rozlohy ostrova, dvě křivky kolonizace pro dvě vzdálenosti ostrova od pevniny. Z jejich průsečíků je možné odhalit, při jakém počtu druhů nastane dynamická rovnováha.
  - V předchozím obrázku je příliš informací, což je někdy na škodu. Ve druhém obrázku necháme jenom jednu z křivek kolonizace, ať je lépe možné sledovat rovnovážnou polohu v závislosti na rozloze ostrova.
  - Dále vyřešíme model pro různé počáteční podmínky. Není překvapení, že vždy se nastolí rovnováha.
  - Nakonec vyřešíme model pro různé hodnoty parametru.

16.3.1 Grafy rychlostí kolonizace a vymírání

```
def kolonizace(N,b=1,beta=1,D=1):
    return b/(D*(N+beta))

def vymirani (N, S=1, a=1, k=1.4):
    return a*N**(k)/S

N = np.linspace(0,10,100)
plt.plot(N,kolonizace(N,b=10, D=1),"k-",
 ↪label="kolonizace, blízký ostrov")
plt.plot(N,kolonizace(N,b=10, D=2,␣
 ↪beta=.5),"k--",label="kolonizace,␣
 ↪vzdálený ostrov")

plt.plot(N,vymirani(N, k=1.5, S = 10),
 ↪"r-",label=r"vymírání, velký ostrov")
plt.plot(N,vymirani(N, k=1.5, S = 2),"r-
 ↪-",label=r"vymírání, malý ostrov")
plt.legend()
ax = plt.gca()
ax.set(
    ylim=(0,10),
    xlim=(0,10),
    title="Dynamika vývoje počtu druhů␣
 ↪na ostrově",
    xlabel="počet druhů",
    ylabel="rychlost kolonizace a␣
 ↪vymírání",
    xticks=[],
    yticks=[]
    );
```



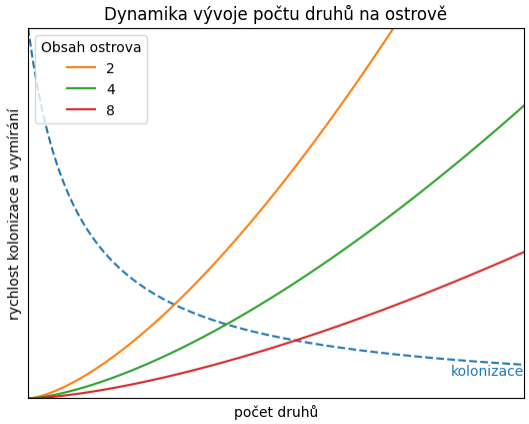
Zkusíme zafixovat rychlost kolonizace a sledovat vymírání jako funkci obsahu ostrova. Měli bychom vidět, že pro menší ostrov je větší rychlost vymírání. Rovnovážný počet druhů najdeme jako průsečík křivky vymírání s čárkovanou křivkou kolonizace. Měli bychom vidět, že první souřadnice průsečíku je pro menší ostrov menší.

```
def kolonizace(N,b=10,beta=1,D=1):
    return b/(D*(N+beta))
def vymirani (N, S=1, a=1, k=1.4):
```

(continues on next page)

(pokračujte na předchozí stránce)

```
    return a*N**(k)/S
meze = [0,10]
obsahy = [2,4,8]

N = np.linspace(*meze,100)
df = pd.DataFrame()
df["N"] = N
for S in obsahy:
    df[S] = vymirani(N, k=1.5, S = S)

fig,ax = plt.subplots(1)
ax.plot(N,kolonizace(N),"--",label="_
 ↪nolegend_")
df.plot(x="N", ax=ax)

ax.legend(title="Obsah ostrova", loc=
 ↪"upper left")
ax.text(N[-1], kolonizace(N[-1]),
 ↪"kolonizace", color="C0", ha="right",␣
 ↪va="top")
ax.set(
    ylim=(0,10),
    xlim=meze,
    title="Dynamika vývoje počtu druhů␣
 ↪na ostrově",
    xlabel="počet druhů",
    ylabel="rychlost kolonizace a␣
 ↪vymírání",
    xticks=[],
    yticks=[]
    );
```



16.3.2 Řešení diferenciální rovnice

Nyní budeme diferenciální rovnici řešit. Použijeme bezrozměrný tvar

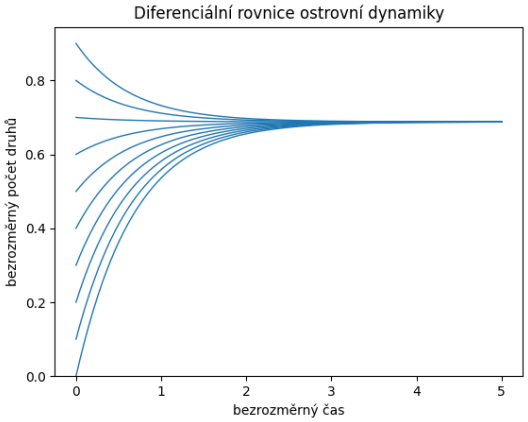
$$\frac{\mathrm{d}n}{\mathrm{d}\tau} = \frac{1}{n+1} - \alpha n^k,$$

kde $\alpha = \frac{a\beta^k}{S}\frac{\beta D}{b}$, $n = \frac{N}{\beta}$ a $\tau = \frac{b}{\beta D}\frac{t}{\beta}$. Tedy parametr α je přímo úměrný obsahu ostrova a nepřímo úměrný vzdálenosti od ostrova, velikost populace měříme v násobcích parametru β a rychlost plynutí času se mění se vzdáleností od ostrova.

16.3.3 Závislost na počáteční podmínce

Využijeme toho, že je možné zadat celou sadu počátečních podmínek a že nemusíme řešit rovnici pro každou počáteční podmínku samostatně. Kromě toho zafixujeme barvu kreslení na první barvu v sadě (`C0`), protože nemá smysl jednotlivá řešení odlišovat barevně. Protože křivek je hodně, změníme tloušťku čáry.

```
pocatecni_podminka = np.arange(0,1,0.1)
meze = [0,5]
def rovnice(t, n, alpha=1, k=1.4):
    return 1/(n+1) - alpha*n**k

t = np.linspace(*meze, 100)  #␣
↪definicni obor pro reseni
reseni = solve_ivp(rovnice, meze,␣
↪pocatecni_podminka, t_eval=t)

fig,ax = plt.subplots(1)
ax.plot(t,reseni.y.T,color="C0",lw=1)
ax.set(
    ylim=(0,None),
    title="Diferenciální rovnice␣
↪ostrovní dynamiky",
    xlabel="bezrozměrný čas",
    ylabel="bezrozměrný počet druhů",
);
```



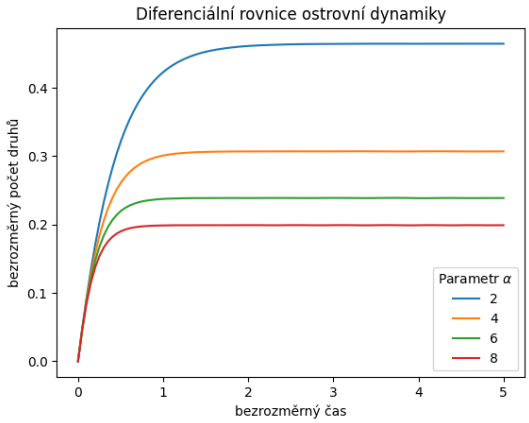
16.3.4 Závislost na parametru α

```
pocatecni_podminka = [0]
meze = [0,5]
alphas = [2,4,6,8]
k = 1.4

t = np.linspace(*meze, 100) # časy ve␣
↪kterých určíme hodnotu řešení
def rovnice(t, n, alpha=1, k=1.4):
    return 1/(n+1) - alpha*n**k
# Trochu jiná taktika určení řešení.␣
↪Budeme ukládat rovnou hodnoty
```

(continues on next page)

(pokračujte na předchozí stránce)

```
# řešení v~uvedených bodech, tj. za␣
↪příkazem pro
# řešení použijeme .y[0]
reseni = [
            solve_ivp(rovnice,
                      meze,
                      pocatecni_
↪podminka,
                      args=(alpha,k),
                      t_eval=t,
                     ).y[0]
            for alpha in alphas
         ]
df = pd.DataFrame(np.array(reseni).T,␣
↪columns=alphas)
df["t"] = t
ax = df.plot(
    x="t",
    title="Diferenciální rovnice␣
↪ostrovní dynamiky",
    xlabel="bezrozměrný čas",
    ylabel="bezrozměrný počet druhů",
)
ax.legend(title=r"Parametr $\alpha$");
```



KAPITOLA 17

Diferenciální rovnice 2

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.integrate import solve_ivp
```

17.1 Exponenciální růst

Namodelujeme exponenciální růst

$$\frac{\mathrm{d}x}{\mathrm{d}t} = kx$$

pro tři různé parametry k. Protože po transformaci do bezrozměrného času $\tau = kt$ má rovnice tvar

$$\frac{\mathrm{d}x}{\mathrm{d}\tau} = x$$

nezávislý na k, ověříme, že všechna řešení po transformaci do bezrozměrného času splývají.

Nejprve růst pro tři různé hodnoty koeficientu k. Vidíme, že čím vyšší je k, tím rychleji velikost populace roste v čase.

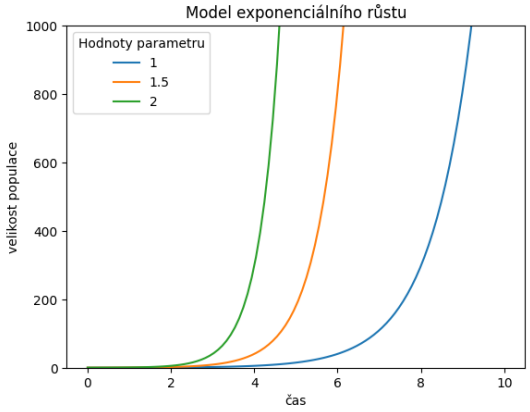
```
### Příprava funkcí a parametrů
pocatecni_podminka = np.array([0.1])  #␣
 ↪počáteční podmínka
meze = np.array([0,10]) # interval, na␣
 ↪kterém hledáme řešení
parametry = [1,1.5,2] # seznam parametrů
n = 100
def rovnice(t, x, k=1):
    return k*x

### Řešení modelu
t=np.linspace(*meze, n)  # definiční␣
 ↪obor, v~těchto bodech budeme hledat␣
 ↪řešení
df = pd.DataFrame()      # tabulka pro␣
 ↪výstup
```

(continues on next page)

(pokračujte na předchozí stránce)

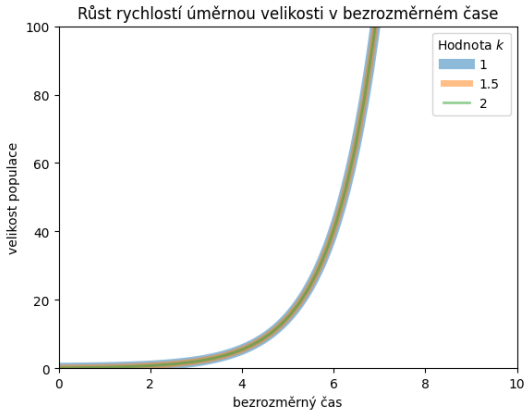
```
df["t"] = t              # sloupec␣
 ↪s~časem

for parametr in parametry:
    reseni = solve_ivp(
                        lambda t,
 ↪x:rovnice(t,x,k=parametr),
                        meze,
                        pocatecni_
 ↪podminka,
                        t_eval=t
                        )
    df[parametr] = reseni.y.T # další␣
 ↪sloupec tabulky
    # lambda funkce viz https://www.
 ↪w3schools.com/python/python_lambda.asp
    # (dočasná nepojmenovaná funkce)

### Vizualizace řešení
ax = df.plot(x="t")
ax.set(
    ylim = (0,1000),
    title = "Model exponenciálního růstu
 ↪",
    xlabel="čas",
    ylabel="velikost populace",
)
plt.legend(title="Hodnoty parametru")
```

```
<matplotlib.legend.Legend at␣
 ↪0x7fac83f0a290>
```



Po transformaci do bezrozměrného času by měly křivky splynout. Abychom to viděli graficky, budeme postupně snižovat jejich tloušťku a nastavíme částečnou průhlednost. To zajistíme volbami `lw` (zkratka za `linewidth`) a `alpha`.

```
for k,lw in zip(parametry,[8,5,2]):
    plt.plot(df["t"]*k,df[k],lw=lw,
 ↪label=k,alpha=0.5)  # Na vodorovne␣
 ↪ose neni cas ale bezrozmerny cas
ax = plt.gca()
ax.set(
    ylim=(0,100),
    xlim=(0,10),
    xlabel="bezrozměrný čas",
    ylabel="velikost populace",
    title="Růst rychlostí úměrnou␣
 ↪velikosti v~bezrozměrném čase"
)
plt.legend(title=r"Hodnota $k$");
```



17.2 Exponenciální růst k vodorovné asymptotě

17.2.1 Úkol

Nakreslete řešení rovnice růstu k vodorovné asymptotě (von Bertalanffyho růst)

$$\frac{\mathrm{d}L}{\mathrm{d}t} = k(L_\infty - L)$$

pro tři různé hodnoty parametru L_∞.

```
# Sem vložte řešení.
```

17.3 Logistický růst

17.3.1 Úkol

Vykreslete řešení logistické rovnice

$$\frac{\mathrm{d}x}{\mathrm{d}t} = rx\left(1 - \frac{x}{K}\right)$$

pro cca deset počátečních podmínek rovnoměrně rozmístěných mezi 0 a K.

```
# Sem vložte řešení. Můžete jej␣
 ↪rozdělit do více buněk, které si sem␣
 ↪vložíte.
```

17.3.2 Logistická rovnice v bezrozměrném tvaru

Nyní si ukážeme, že v určitém smyslu stačí bez újmy na obecnosti studovat případ $K = 1$ a $r = 1$. Na rovnici

$$\frac{\mathrm{d}y}{\mathrm{d}\tau} = y(1 - y)$$

se totiž logistická rovnice transformuje při přechodu od rovnice

$$\frac{\mathrm{d}x}{\mathrm{d}t} = rx\left(1 - \frac{x}{K}\right)$$

s použitím substituce $y = \frac{x}{K}$ a $\tau = rt$. Najdeme $y(\tau)$ jako řešení bezrozměrné rovnice s jednotkovými parametry a $x(t)$ jako řešení rovnice s náhodnými parametry r a K. Poté ukážeme na několika numerických hodnotách, že platí

$$y(t) = K\, x(rt).$$

```
pocatecni_podminka = np.array([0.1])
meze = np.array([0,10])
def rovnice(t, x, r=1, K=1):
    return r*x*(1-x/K)

# r a K~budou náhodná čísla z~intervalu␣
 ↪(0,0.2) a (0,10).
K,r = np.array([10,0.2])*np.random.
 ↪random(2)
print(f"r={r}, K={K}")
t = np.linspace(*meze,12)

reseni_plne_rovnice = solve_ivp(
                    lambda t,x:rovnice(t,
 ↪x,r=r,K=K),
                    meze,
                    pocatecni_podminka,
                    t_eval=t
                    )

reseni_bezrozmerne_rovnice = solve_ivp(
                    rovnice,
                    r*meze,  # převod␣
 ↪mezí na bezrozměrný čas
                    pocatecni_podminka/K,
 ↪  # převod na bezrozměrnou velikost␣
 ↪populace
                    t_eval=r*t #␣
 ↪vyhodnoceni v~bezrozmernem case
                    )

df = pd.DataFrame()
df["t"] = t
df["plnohodnotna"] = reseni_plne_
 ↪rovnice.y[0]
bezrozmerna_velikost = reseni_
 ↪bezrozmerne_rovnice.y[0]
velikost_populace = K*bezrozmerna_
 ↪velikost
df["bezrozmerna"] = velikost_populace
df["rozdil"] = np.abs(df["bezrozmerna"]-
 ↪df["plnohodnotna"])
df["relativni rozdil"] = df["rozdil"]/
 ↪df["plnohodnotna"]
df
```

```
r=0.0778412556259561, K=3.
 ↪9481196827904155
```

```
            t  plnohodnotna ␣
 ↪bezrozmerna        rozdil ␣
 ↪relativni rozdil
0    0.000000      0.100000     0.
 ↪100000  0.000000e+00      0.
 ↪000000e+00
1    0.909091      0.107134     0.
 ↪107134  5.496270e-10      5.
 ↪130282e-09
2    1.818182      0.114762     0.
 ↪114762  8.362731e-09      7.
```

(continues on next page)

(pokračujte na předchozí stránce)

```
 ↪287050e-08
3    2.727273      0.122915     0.
 ↪122915  5.630983e-08      4.
 ↪581207e-07
4    3.636364      0.131628     0.
 ↪131627  1.086148e-07      8.
 ↪251679e-07
5    4.545455      0.140935     0.
 ↪140935  1.307709e-07      9.
 ↪278804e-07
6    5.454545      0.150875     0.
 ↪150875  1.025540e-07      6.
 ↪797292e-07
7    6.363636      0.161486     0.
 ↪161486  1.801810e-08      1.
 ↪115769e-07
8    7.272727      0.172809     0.
 ↪172810  1.145049e-07      6.
 ↪626079e-07
9    8.181818      0.184888     0.
 ↪184888  2.724051e-07      1.
 ↪473350e-06
10   9.090909      0.197767     0.
 ↪197767  4.187947e-07      2.
 ↪117617e-06
11  10.000000      0.211492     0.
 ↪211493  5.025078e-07      2.
 ↪376011e-06
```

```
df.rozdil.max()
```

```
5.025077921827492e-07
```

```
df["relativni rozdil"].max()
```

```
2.376010567565709e-06
```

```
np.isclose(df["bezrozmerna"],df[
 ↪"plnohodnotna"], rtol=0.0001).all()
```

```
True
```

KAPITOLA 18

Diferenciální rovnice 3

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.integrate import solve_ivp
```

18.1 Lov v logistické rovnici

18.1.1 Konstantní užitek

V logistické rovnici nakreslíme pro tři různé intenzity lovu průběh řešení. Prorovnejte následující kód s kódem pro kreslení řešení jedné sady pro jednu intenzitu lovu.

Zpravidla netriviální kód nenapíšeme napoprvé, ale musíme příkazy ladit. V následujícím jsou rozděleny fáze řešení rovnice a vykreslení řešení. Pro potřeby spouštění je po odladění vhodné buňky sloučit.

- Zkuste si v následující buňce rozdělit kód do dvou různých buněk. Tedy přepnete se do editace, najdete vhodný řádek a v menu vyberete „Edit“ a „Split Cell“.
- Poté zkuste buňky co se mají spouštět společně spojit. V příkazovém modu buňky označte (například shift + šipka nahoru nebo dolů) a stisknout velké M, tj. Shift + M. Pozor, pokud byste stiskli malé „m“, buňka by se změnila na Markdown buňku s textem. Zpět na buňku s kódem je klávesa „y“.

```
pocatecni_podminka = np.linspace(0.1,1.
 ↪2,50)
meze = [0,10]
t = np.linspace(*meze,100)

def destrukce_populace(t,x):  # Pokud x␣
 ↪klesne na nulu, zastavíme výpočet
    return x
destrukce_populace.terminal = True
```

(continues on next page)

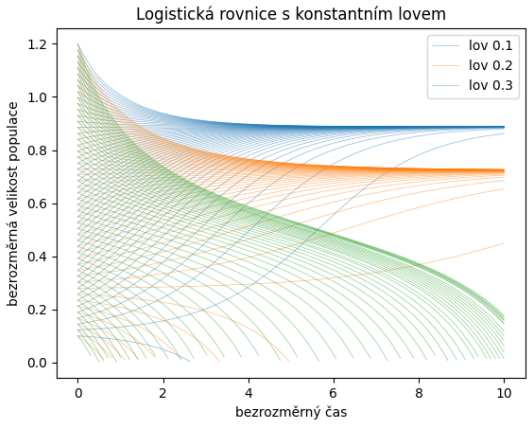
(pokračujte na předchozí stránce)

```
def rovnice(t, x, r=1, K=1, h=0.15):
    return r*x*(1-x/K)-h

lovy = [0.1,0.2,0.3]
# Pro různé počáteční podmínky se bude␣
 ↪lišit interval,
# na kterém algoritmus najde řešení.␣
 ↪Proto nemůžeme data
# shrnout do jedné tabulky.␣
 ↪Alternativou je tabulka s~
# nedefinovanými hodnotami, viz
# https://robert-marik.github.io/dmp/
 ↪snippety/tabulky_none.html
# a https://robert-marik.github.io/dmp/
 ↪snippety/multiheader.html
reseni = [
        [ solve_ivp(
                    lambda t,x:rovnice(t,
 ↪x,h=h),
                    meze,
                    [pp],
                    t_eval=t,
                    events=destrukce_
 ↪populace,  # zastavení výpočtu při␣
 ↪poklesu populace na nulu
                    )
          for pp in pocatecni_podminka]
        for h in lovy]

# GRAFICKA PREZENTACE VYSLEDKU
fig,ax = plt.subplots()
for i,r in enumerate(reseni):
    for res in r:
        ax.plot(res.t,res.y[0], color=f
 ↪"C{i}", alpha=0.5, label=f"lov
 ↪{lovy[i]}", lw=0.5)
ax.set(
  title="Logistická rovnice␣
 ↪s~konstantním lovem",
  xlabel="bezrozměrný čas",
  ylabel="bezrozměrná velikost populace"
```

(continues on next page)

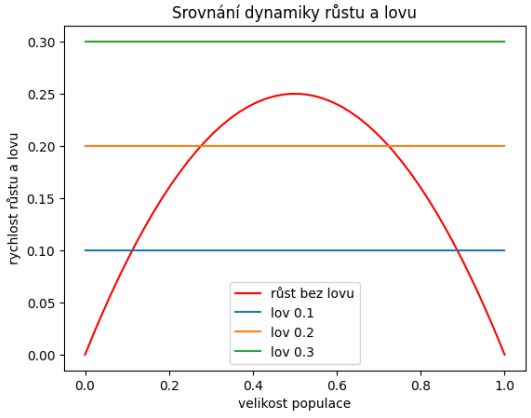
(pokračujte na předchozí stránce)

```
  );

# Návod jak seskupit položky legendy je␣
 ↪na https://stackoverflow.com/
 ↪questions/26337493/pyplot-combine-
 ↪multiple-line-labels-in-legend
handles, labels = ax.get_legend_handles_
 ↪labels()
labels, ids = np.unique(labels, return_
 ↪index=True)
handles = [handles[i] for i in ids]
plt.legend(handles, labels);
```




```
fig,ax = plt.subplots()
x = np.linspace(0,1)
plt.plot(x,rovnice(0, x,h=0),label=
 ↪"růst bez lovu",color='r')
for h in lovy:
    plt.plot(x,h+0*x,label=f"lov {h}")
ax.set(
    xlabel="velikost populace",
    ylabel="rychlost růstu a lovu",
    title="Srovnání dynamiky růstu a␣
 ↪lovu"
)
plt.legend()
```

```
<matplotlib.legend.Legend at␣
 ↪0x7f9e725f6bd0>
```



18.1.2 Konstantní úsilí

Modifikujte předchozí kód s vykreslením časového vývoje populace vystavené lovu. Lov konstantní intenzitou nahraďte lovem s konstantním úsilím. Zkuste nejprve minimální úprava kódu. Bez ohledu na efektivitu, vyjděte z předchozího a snažte se co nejméně modifikovat výchozí kód. Poté si prostudujte elegantnější přístup využívající toho, že žádné řešení neskončí v konečném čase.

```
# sem napiste reseni
```

U konstantního úsilí není problém s tím, že by některá řešení končila dříve. Proto může být programový kód kratší a čistý. Například nemusíme pracovat s vnořenými cykly a můžeme příkazu `solve_ivp` poslat současně všechny počáteční podmínky. Pokusme se o to.

```
pocatecni_podminka = np.linspace(0.1,1.
 ↪2,51).round(3)
meze = [0,10]
t = np.linspace(*meze,100)

def rovnice(t, x, r=1, K=1, h=0.15):
    return r*x*(1-x/K)-h*x

lovy = [0.1,0.3,0.6]

### Definice tabulky s~víceúrovňovými␣
 ↪nadpisy sloupců, MultiIndex
### https://pandas.pydata.org/docs/user_
 ↪guide/advanced.html
iterables = [["data"],lovy,pocatecni_
 ↪podminka]
my_index = pd.MultiIndex.from_
 ↪product(iterables, names=['typ','lov',
 ↪ 'poč.podm.'])
df = pd.DataFrame(columns=my_index)

df["čas"] = t

for h in lovy:
    r = solve_ivp(
            lambda t,x:rovnice(t,x,
 ↪h=h),
            meze,
            pocatecni_podminka,
            t_eval=t,
            ).y.T
    df[[("data",h,i) for i in pocatecni_
 ↪podminka]]=pd.DataFrame(r)

df.T
```

```
                        0         1  ␣
 ↪       2         3         4       ␣
 ↪ 5   \
typ  lov poč.podm.
data 0.1 0.1         0.100  0.108371 ␣
 ↪0.117336  0.126920  0.137151  0.
```

(continues on next page)

(pokračujte na předchozí stránce)

```
↪148054
        0.122      0.122  0.131909 ␣
↪0.142472  0.153707  0.165637  0.
↪178276
        0.144      0.144  0.155339 ␣
↪0.167371  0.180106  0.193555  0.
↪207723
        0.166      0.166  0.178662 ␣
↪0.192036  0.206123  0.220922  0.
↪236423
        0.188      0.188  0.201879 ␣
↪0.216470  0.231766  0.247752  0.
↪264403
...                   ...       ... ␣
↪     ...       ...       ...      ␣
↪...
    0.6 1.134      1.134  1.057181 ␣
↪0.991996  0.936033  0.888480  0.
↪848319
        1.156      1.156  1.075507 ␣
↪1.007379  0.949009  0.899631  0.
↪858199
        1.178      1.178  1.093759 ␣
↪1.022631  0.961806  0.910585  0.
↪867910
        1.2        1.200  1.111938 ␣
↪1.037750  0.974419  0.921339  0.
↪877452
čas                0.000  0.101010 ␣
↪0.202020  0.303030  0.404040  0.
↪505051

                           6        ␣
↪7         8         9   ...       ␣
↪90  \
typ  lov poč.podm.                  ␣
↪                        ...
data 0.1 0.1        0.159650  0.
↪171950  0.184964  0.198694  ...  0.
↪898016
        0.122      0.191634  0.
↪205712  0.220505  0.236000  ...  0.
↪898404
        0.144      0.222605  0.
↪238190  0.254458  0.271383  ...  0.
↪898676
        0.166      0.252609  0.
↪269455  0.286929  0.304989  ...  0.
↪898878
        0.188      0.281690  0.
↪299574  0.318011  0.336948  ...  0.
↪899034
...                       ...       ..
↪.       ...       ...  ...       ..
↪.
    0.6 1.134      0.814316  0.
↪785022  0.758774  0.733722  ...  0.
↪407263
        1.156      0.823389  0.
↪793589  0.766905  0.741191  ...  0.
↪407361
```

(continues on next page)

(pokračujte na předchozí stránce)

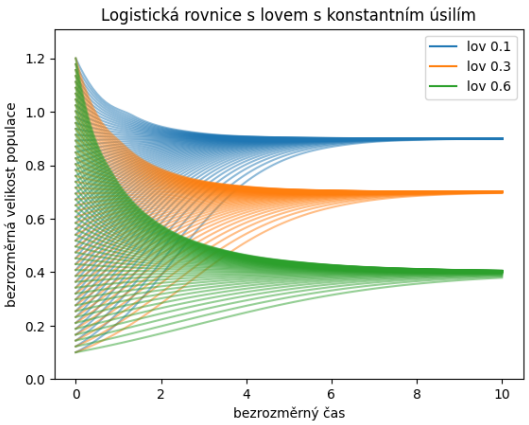
```
        1.178      0.832358  0.
↪802137  0.775090  0.748730  ...  0.
↪407459
        1.2        0.841232  0.
↪810687  0.783359  0.756369  ...  0.
↪407558
čas                0.606061  0.
↪707071  0.808081  0.909091  ...  9.
↪090909

                           91       ␣
↪92        93        94        95  \
typ  lov poč.podm.
data 0.1 0.1        0.898182  0.
↪898329  0.898459  0.898573  0.
↪898675
        0.122      0.898538  0.
↪898656  0.898760  0.898852  0.
↪898934
        0.144      0.898787  0.
↪898885  0.898971  0.899047  0.
↪899115
        0.166      0.898972  0.
↪899055  0.899128  0.899192  0.
↪899250
        0.188      0.899115  0.
↪899186  0.899249  0.899304  0.
↪899354
...                       ...       ..
↪.       ...       ...       ...
    0.6 1.134      0.406970  0.
↪406690  0.406421  0.406163  0.
↪405915
        1.156      0.407065  0.
↪406780  0.406508  0.406246  0.
↪405995
        1.178      0.407159  0.
↪406871  0.406594  0.406329  0.
↪406075
        1.2        0.407253  0.
↪406961  0.406681  0.406412  0.
↪406155
čas                9.191919  9.
↪292929  9.393939  9.494949  9.
↪595960

                           96       ␣
↪97        98         99
typ  lov poč.podm.
data 0.1 0.1        0.898768  0.
↪898856  0.898940   0.899027
        0.122      0.899008  0.
↪899078  0.899147   0.899216
        0.144      0.899177  0.
↪899235  0.899291   0.899349
        0.166      0.899302  0.
↪899351  0.899399   0.899448
        0.188      0.899399  0.
↪899441  0.899482   0.899524
...                       ...       ..
↪.       ...       ...
```

(continues on next page)

(pokračujte na předchozí stránce)

```
      0.6 1.134       0.405678  0.
↪405450  0.405232   0.405022
         1.156       0.405755  0.
↪405524  0.405303   0.405090
         1.178       0.405831  0.
↪405597  0.405373   0.405157
         1.2         0.405908  0.
↪405671  0.405443   0.405225
čas                  9.696970  9.
↪797980  9.898990  10.000000

[154 rows x 100 columns]
```

```
fig,ax = plt.subplots()
for i in range(len(lovy)): # tři čáry␣
↪mimo obrázek kvůli legendě
    ax.plot([0,1],[-1,-1],label=f"none")
for i,h in enumerate(lovy):
    ax.plot(df["čas"],df["data"][h],␣
↪color=f"C{i}", alpha=0.5)
ax.set(
  title="Logistická rovnice s~lovem␣
↪s~konstantním úsilím",
  xlabel="bezrozměrný čas",
  ylabel="bezrozměrná velikost populace
↪",
  ylim=[0,None]
  )

plt.legend([f"lov {lov}" for lov in␣
↪lovy]);
```



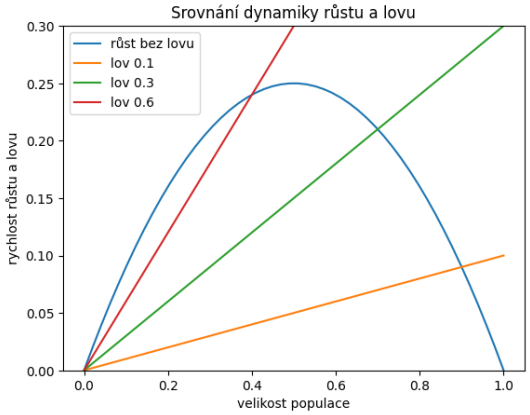

```
fig,ax = plt.subplots()
x = np.linspace(0,1)
plt.plot(x,rovnice(0, x,h=0),label=
↪"růst bez lovu")
for a in lovy:
    plt.plot(x,a*x,label=f"lov {a}")
ax.set(
    xlabel="velikost populace",
    ylabel="rychlost růstu a lovu",
    title="Srovnání dynamiky růstu a␣
↪lovu",
    ylim=(0,0.3)
```

(continues on next page)

(pokračujte na předchozí stránce)

```
)
plt.legend()
```

```
<matplotlib.legend.Legend at␣
↪0x7f9e70cd5950>
```



18.2 Alleeho efekt

Nakreslete model řešení rovnice modelující lov konstantním úsilím v populaci s Alleeho efektem. Můžete uvažovat slabý Aleeho efekt (pro malé velikosti populace se dynamika růstu výrazně zpomalí) nebo silný Aleeho efekt (pro malé velikosti populace vymírá). Můžete použít například rovnici

$$\frac{\mathrm{d}x}{\mathrm{d}t} = rx^k\left(1-\frac{x}{K}\right) - ax, \quad k > 1$$

pro slabý nebo

$$\frac{\mathrm{d}x}{\mathrm{d}t} = rx\left(1-\frac{x}{K}\right)\left(\frac{x}{A}-1\right) - ax, \quad 0 < A < K$$

pro silný Alleeho efekt. Nejprve si nakreslete křivky růstu a lovu, pro vytipování vhodných numerických hodnot pro parametry a poté nakreslete řešení diferenciální rovnice pro různé intenzity lovu a různé počáteční podmínky.

```
# sem napiste kod pro vykreslení krivek␣
↪rustu a lovu
```

```
# sem napiste kod pro vykreslení reseni␣
↪diferenciální rovnice
```

18.3 Populace pod predačním tlakem

Vykreslete model pro populaci pro predačním tlakem https://robert-marik.github.io/dmp/prednaska/05.html#populace-pod-predacnim-tlakem. Použijte bezrozměnrou formulaci, tj. rovnici

$$\frac{\mathrm{d}x}{\mathrm{d}\tau} = \alpha\left(1-\frac{x}{\beta}\right)x - \frac{x^2}{x^2+1}.$$

Tři dvojice hodnot pro α a β můžete použít z https://robert-marik.github.io/dmp/prednaska/05.html#rustove-krivky.

```
# sem napiste reseni
```

KAPITOLA 19

LaTeX

V tomto cvičení se naučíme psát texty obsahující matematiku. Pravidla jsou následující.

1. Každá věta začíná velkým písmenem, končí tečkou a nikdy není odstavec rozdělen uvnitř věty. Aby bylo snadné toto pravidlo dodržet nezačínáme nikdy větu matematickým výrazem. Interpunkci používáme dle pravidel českého jazyka a to i v případě, že matematický výraz stojí na samostatném řádku.
   - Špatně: k je konstanta úměrnosti.
   - Správně: Konstanta k je konstanta úměrnosti.
   - Špatně: Obsah kruhu o poloměru r je dán vztahem
     $$S = \pi r^2$$
   - Správně: Obsah kruhu o poloměru r je dán vztahem
     $$S = \pi r^2.$$
   - Špatně: Obsah kruhu je dán vztahem
     $$S = \pi r^2$$
     kde r je poloměr kruhu.
   - Správně: Obsah kruhu je dán vztahem
     $$S = \pi r^2,$$
     kde r je poloměr kruhu.
2. Každá matematická proměnná se označuje jako matematický výraz. Editor poté použije automaticky předvolený font, zpravidla matematickou kurzívu nebo kurzívu.
   - V MS Word je toto pod klávesovou zkratkou `Alt=`.
   - V systému LaTeX se začátek a konec matematického výrazu vyznačuje dle interpretru dolary, znaky `\(`, `\)`, `\[` a `\]` nebo speciálními tagy (Wikipedie).
3. Důležité matematické vzorce píšeme na samostatný řádek a dle typografie buď centrujeme nebo zarovnáváme na levý okraj. Přitom nekončíme odstavec uvnitř první věty (pravidlo 1). Viz příklad v prvním bodě. V LaTeXu pro vzorec na samostatném řádku používáme dvojdolar nebo hranatou závorku, pro vzorec uvnitř řádku jednoduchý dolar a kulatou závorku. Ve Wordu vzorec na samostatný řádek umístíme pravým tlačítkem a výběrem odpovídající položky v menu.
4. Matematické funkce zapisujeme jiným fontem než proměnné. Toto zařídí software automaticky, pokud dodržujeme nastavená pravidla.
   - Špatně: `sin(x)+cos(x)` dává $sin(x) + cos(x)$
   - Správně: `\sin(x)+\cos(x)` dává $\sin(x) + \cos(x)$

Další výrazy zapisujeme smluvenými značkami. Například kód pro derivaci $\frac{\mathrm{d}x}{\mathrm{d}t}$ si zobrazíte kliknutím na tento text.

KAPITOLA 20

Maticové modely

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#from scipy.integrate import solve_ivp
from scipy import optimize
np.set_printoptions(suppress=True)
```

Při práci s maticemi musíme dávat pozor, že když proměnnou uložíme do jiné proměnné a novou proměnnou poté modifikujeme, modifikuje se i původní objekt. To proto, že nevzniká nová matice, ale jenom nový odkaz na matici původní.

Vyzkoušejte si následující kód.

```
a = np.matrix([1,2,3,4]) # vytvoření␣
 ↪matice a
b = a  # uložení do matice b
b[0,3] = -100  # změna jednoho prvku␣
 ↪matice b
print(a) # test jestli se změnila␣
 ↪matice a
print(b) # test jestli se změnila␣
 ↪matice b
```

```
[[   1    2    3 -100]]
[[   1    2    3 -100]]
```

Pokud je výše popsané chování nežádoucí, použijeme `copy`.

```
a = np.matrix([1,2,3,4]) # vytvoření␣
 ↪matice a
b = a.copy()   # uložení KOPIE do␣
 ↪matice b
b[0,3] = -100  # změna jednoho prvku␣
 ↪matice b
print(a) # test jestli se změnila␣
 ↪matice a
print(b) # test jestli se změnila␣
 ↪matice b
```

```
[[1 2 3 4]]
[[   1    2    3 -100]]
```

20.1 Leslieho model

Nejprve prozkoumáme Lesliseho model populace tuleňů z přednášky. Prvotní informace se týká vývoje populace. Rychlou představu soi uděláme z grafu sledujícího vývoj jednotlivých věkových skupin v čase.

```
L = np.matrix([
    0, 1.26, 2.0,
    0.614, 0, 0,
    0, 0.808, 0.808
]).reshape(3,3)
L
```

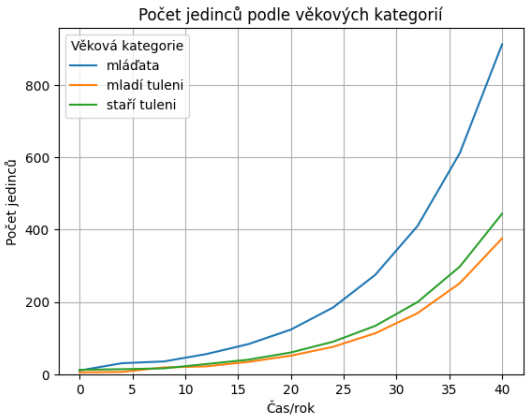
```
matrix([[0.   , 1.26 , 2.   ],
        [0.614, 0.   , 0.   ],
        [0.   , 0.808, 0.808]])
```

```
N = 10
X = np.zeros((3,N+1))
X[:,0] = [10,5,12]
for i in range(N):
    X[:,[i+1]] = L*X[:,[i]]

fig, ax = plt.subplots(1)
cas = np.linspace(0,N,N+1) * 4 # časový␣
 ↪krok v~modelu jsou čtyři roky
ax.plot(cas,X.T)
ax.legend(
    ["mláďata","mladí tuleni","staří␣
 ↪tuleni"],
    title="Věková kategorie"
    )
ax.set(
    ylim=(0,None),
```

(continues on next page)

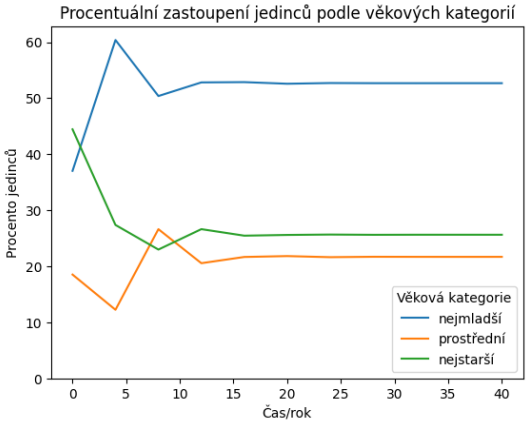
(pokračujte na předchozí stránce)

```
    title="Počet jedinců podle věkových␣
 ↪kategorií",
    ylabel="Počet jedinců",
    xlabel="Čas/rok"
    )

ax.grid();
```



Ověříme, že věková struktura konverguje ke stálému složení populace.

```
#https://stackoverflow.com/questions/
 ↪61888227/numpy-2d-array-get-
 ↪percentage-of-total
soucet_sloupcu = X.sum(axis=0)
podil = X/soucet_sloupcu * 100
plt.plot(cas,podil.T)
ax = plt.gca()
ax.set(
    ylim=(0,None),
    title="Procentuální zastoupení␣
 ↪jedinců podle věkových kategorií",
    ylabel="Procento jedinců",
    xlabel="Čas/rok"
    )
ax.legend(
    ["nejmladší","prostřední","nejstarší
 ↪"],
    title="Věková kategorie"
    );
```



Numerické hodnoty určující procentuální zastoupení jednotlivých věkových skupin získáme z konce simulace.

```
podil[:,-1]
```

```
array([52.68328969, 21.68662839, 25.
 ↪63008192])
```

Lesliho matice má (za předpokladu, že alespoň dva po sobě jdoucí koeficienty v prvním řádku matice jsou nenulové) dominantní reálnou vlastní hodnotu. Vektor odpovídající této vlastní hodnotě určuje zastoupení jednotlivých věkových tříd. Tuto informaci můžeme porovnat s již zjištěnou strukturou populace. Obecně mohou být vlastní čísla a vlastní vektory i komplexní.

```
v,P = np.linalg.eig(L)
print("Vlastní čísla: ",v)
print("Vlastní vektory jsou sloupce␣
 ↪matice\n",P)
```

```
Vlastní čísla:  [-0.34182332+0.
 ↪35954975j -0.34182332-0.35954975j ␣
 ↪1.49164663+0.j        ]
Vlastní vektory jsou sloupce matice
 [[ 0.38392628-0.40383611j  0.
 ↪38392628+0.40383611j  0.84331389+0.
 ↪j         ]
 [-0.68962744+0.j          -0.
 ↪68962744-0.j          0.34712962+0.
 ↪j         ]
 [ 0.44144739+0.1380406j   0.
 ↪44144739-0.1380406j   0.4102715 +0.
 ↪j         ]]
```

Odfiltrujeme komplexní čísla. Kvůli numerickému zaokrouhlování nemá smysl u komplexních čísel testovat, zda je imaginární část přesně nulová. Namísto toho zvolíme jistý práh, od kterého budeme imaginární část považovat za nulovou.

```
vh,vs = [ [i.real,j.real] for i,j in␣
 ↪zip(v,P.T) if np.abs(i.imag)<1e-2 ][0]
vs = vs.A1 # převod na vektor
vs = vs/sum(vs) # normování aby součet␣
 ↪komponent byl jedna
print("Reálná vlastní hodnota/hodnoty␣
 ↪je/jsou ",vh)
print("Příslušný vlastní směr/směry je/
 ↪jsou ", vs)
```

```
Reálná vlastní hodnota/hodnoty je/
 ↪jsou  1.4916466348833581
Příslušný vlastní směr/směry je/jsou␣
 ↪ [0.52683575 0.2168591  0.25630515]
```

```
l_index = np.argmax(v)
l = max(v).real
l
```

```
1.4916466348833581
```

Poznámka: Při porovnávání komplexních čísel používá knihovna NumPy reálnou komponentu. Toto chování vyhovuje u vlastních čísel v případě, kdy víme, že dominantní vlastní hodnota je reálná. Leslieho matice je představitelem takové úlohy, kdy jistotu reálné vlastní hodnoty zpravidla máme.

```
# U~komplexních čísel neexistuje relace␣
↪uspořádání podle velikosti.
# Pro porovnávání se v~knihovně NumPy␣
↪používá reálná část.
np.max([0+300j, 2+0j, 1+4j])
```

```
(2+0j)
```

20.1.1 Parametr pro zastavení růstu

Budeme hledat, jak je potřeba změnit parametry populace tak, aby se růst populace zastavil. K zastavení vybereme vhodný parametr. Například parametr ve druhém řádku a prvním sloupci. Tento parametr udává, jaká je pravděpodobnost, že se nejmladší věková kategorie dožije přestupu do starší kategorie. Budeme se tedy snažit lovit nejmladší jedince v populaci. Otázkou je, jak intenzivně musíme lovit, aby se růst populace zastavil.

```
# musíme vytvořit kopii matice, aby se␣
↪prvky původní matice neměnily
L_fix = L.copy()
# Potřebujeme, aby největší vlastní␣
↪číslo bylo rovno jedné.
# K~tomu nadefinujeme funkci, která nám␣
↪toto vlastní číslo vypočítá
# v~závislosti na tom, jak se změní␣
↪prvek, se kterým manipulujeme.
def nejvetsi_vlastni_cislo(x, M):
    M[1,0] = x
    v = np.linalg.eigvals(M)
    return max(v).real
# Nakonec určíme, pro jakou hodnotu je␣
↪vlastní číslo rovno jedné.
oprava = optimize.fsolve(lambda␣
↪x:nejvetsi_vlastni_cislo(x, L_fix)-1,
↪0.6)
L_fix
```

```
matrix([[0.        , 1.26      , 2.
↪       ],
        [0.10334137, 0.        , 0.
↪       ],
        [0.        , 0.808     , 0.
↪808     ]])
```

(continues on next page)

(pokračujte na předchozí stránce)

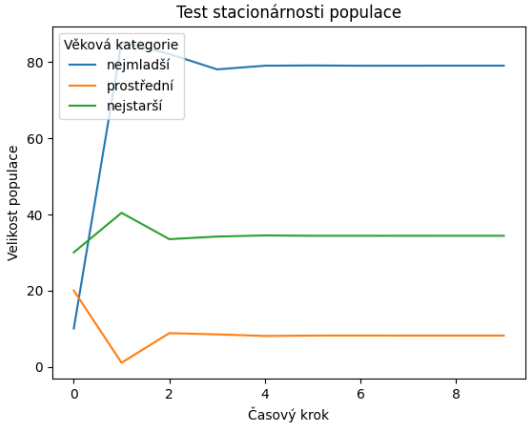
Kontrola vlatních čísel. Dominantní vlastní číslo musí mít reálnou část jednotkovou a komplexní část nulovou. Tedy `1+0j`.

```
np.linalg.eigvals(L_fix)
```

```
array([-0.096+0.22928993j, -0.096-0.
↪22928993j,  1.   +0.j        ])
```

Rychlá vizuální kontrola pro ty, co se nechtějí spoléhat na vlastní čísla. Také pomůže pro odhad, jak rychle se věkové složení ustálí.

```
ini = np.matrix([10,20,30]).reshape(3,1)
k = 10
model = np.zeros([3,k])
model[:,[0]] = ini
for i in range(k-1):
    model[:,[i+1]] = L_fix * model[:,
↪[i]]
plt.plot(model.T)
plt.xlabel("Časový krok")
plt.ylabel("Velikost populace")
plt.legend(
    ["nejmladší","prostřední","nejstarší
↪"],
    title="Věková kategorie"
    )
plt.title("Test stacionárnosti populace
↪");
```



20.1.2 Matice sensitivity a elasticity

Výpočet matice elasticity pomocí levých a pravých vlastních vektorů. Pro násobení matic po složkách (Hadamardův součin) převedeme matice na dvourozměrná pole.

```
W = P[:,l_index].real # pravý sloupcový␣
↪vlastni vektor příslušný maximální␣
↪vlastní hodnotě
V = (P**(-1))[l_index,:].real # levý␣
↪řádkový vlastní vektor příslušný␣
↪maximální vlastní hodnotě
matice_elasticity = np.matrix(np.
↪array(V.T*W.T)*np.array(L)/l)
matice_elasticity
```

```
matrix([[0.        , 0.10157076, 0.
↪19054952],
        [0.29212028, 0.        , 0. ␣
↪       ],
        [0.        , 0.19054952, 0.
↪22520993]])
```

Výpočet matice elasticity z definice pomocí parciálních derivací a centrální diference. Jedná se o výpočet přímo z definice. Výpočet je méně elegantní než výpočet z předchozího pole, ale nespoléhá se například na vztah mezi levými a pravými vlastními vektory. Podobné taktiky, kdy stejnou veličinu počítáme dvěma různými způsoby, se často používají k tomu, abychom ověřili, že programový kód neobsahuje logické chyby a že počítá to, co potřebujeme.

```
h = 0.01
matice = np.zeros([3,3])
matice
for f1 in range(3):
    for f2 in range(3):
        if L[f1,f2] == 0:
            matice[f1,f2] = 0
        else:
            L2 = L.copy()
            L3 = L.copy()
            L2[f1,f2] = L2[f1,f2] + h
            L3[f1,f2] = L3[f1,f2] - h
            v2 = np.linalg.eigvals(L2).
↪max().real
            v3 = np.linalg.eigvals(L3).
↪max().real
            matice[f1,f2] = ((v2-v3)/
↪(2*h))*L[f1,f2]/l
matice = np.matrix(matice)
matice
```

```
matrix([[0.        , 0.10157068, 0.
↪19055025],
        [0.29212876, 0.        , 0. ␣
↪       ],
        [0.        , 0.19055404, 0.
```

(continues on next page)

(pokračujte na předchozí stránce)

```
↪22521132]])
```

```
np.sum(matice_elasticity)
```

```
1.0
```

```
np.sum(matice_elasticity, axis=1)
```

```
matrix([[0.29212028],
        [0.29212028],
        [0.41575944]])
```

```
np.sum(matice_elasticity,axis=0)
```

```
matrix([[0.29212028, 0.29212028, 0.
↪41575944]])
```

Levý vlastní vektor je **reprodukční hodnota** jednotlivých věkových kategorií. Udává očekávaný počet potomků na jedince v dané věkové kategorii a v kategoriích následujících. Počty jsou vyjádřeny relativně vzhledem k nejmladší kategorii, která má reprodukční hodnotu rovnu jedné. Přitom se nejedná o prostý součet, ale je zde zohledněn růst populace, tedy jestli se budoucí potomci rodí do větší či menší populace (viz Tkadlec strana 121).

```
V/V[0,0]
```

```
matrix([[1.        , 2.42939191, 2.
↪92548796]])
```

20.2 Model jelena evropského (red deer, *Cervus elaphus*)

Model je představen a studován ve výukovém materiálu https://www.youtube.com/watch?v=cMQd-okvS_M Populace laní jelena evropského je rozdělena na rok staré laně, dva roky staré laně a starší laně. Projekční matice populace je následující.

```
L = np.matrix([0, 0.2, 0.26,
               0.93, 0, 0,
               0, 0.97, 0.91]).reshape(3,
↪3)
L
```

```
matrix([[0.  , 0.2 , 0.26],
        [0.93, 0.  , 0.  ],
        [0.  , 0.97, 0.91]])
```

Nejprve vypočteme vlastní čísla a najdeme dominantní vlastní číslo.

```
vlastni_cisla, vlastni_vektory = np.
 ↪linalg.eig(L)
maximum_index = np.argmax(vlastni_cisla)
vlastni_cisla[maximum_index]
```

```
(1.1265483882211464+0j)
```

Pro jednoduché seznámení se s modelem je možné snížit jednu komponentu například o deset procent a sledovat odezvu, jak moc se změní dominantní vlastní hodnota.

```
L_redukce_porodnosti = L.copy()
L_redukce_porodnosti[0,2] = 0.9*L_
 ↪redukce_porodnosti[0,2]
np.linalg.eigvals(L_redukce_porodnosti).
 ↪max()
```

```
(1.111253336399596+0j)
```

```
L_redukce_prezivani = L.copy()
L_redukce_prezivani[2,2] = 0.9*L_
 ↪redukce_prezivani[2,2]
np.linalg.eigvals(L_redukce_prezivani).
 ↪max()
```

```
(1.0658700220796447+0j)
```

Detailnější analýzu poskytne matice elasticity.

```
W = vlastni_vektory[:,maximum_index]
V = (vlastni_vektory**(-1))[maximum_
 ↪index,:]

matice_sensitivity = np.around(V.T*W.T,
 ↪decimals=5).real
matice_elasticity = (np.array(matice_
 ↪sensitivity)*np.array(L)/max(vlastni_
 ↪cisla)).real
matice_elasticity
```

```
array([[0.        , 0.02275801, 0.
 ↪13252391],
       [0.15528228, 0.        , 0.
 ↪        ],
       [0.        , 0.13252223, 0.
 ↪55690577]])
```

```
matice_elasticity.sum(axis=0)
```

```
array([0.15528228, 0.15528024, 0.
 ↪68942968])
```

Reprodukční hodnotu můžeme určit z levého vlastního vektoru příslušného dominantní vlastní hodnotě.

```
(V/V[0,0]).real
```

```
matrix([[1.        , 1.21134235, 1.
 ↪20065544]])
```

Pro kontrolu můžeme vypočítat komponenty matice elasticity z definice, kdy parciální derivaci nahradíme centrální diferencí.

```
h = 0.001
matice = np.zeros([3,3])
matice
for f1 in [0,1,2]:
    for f2 in [0,1,2]:
        if L[f1,f2] == 0:
            matice[f1,f2] = 0
        else:
            L2 = L.copy()
            L3 = L.copy()
            L2[f1,f2] = L2[f1,f2] + h
            L3[f1,f2] = L3[f1,f2] - h
            v2,P2 = np.linalg.eig(L2)
            v3,P3 = np.linalg.eig(L3)
            matice[f1,f2] = ((max(v2)-
 ↪max(v3))/(2*h)*L[f1,f2]/max(vlastni_
 ↪cisla)).real
matice = np.matrix(matice)
matice
```

```
matrix([[0.        , 0.02275818, 0.
 ↪13252514],
        [0.15528314, 0.        , 0.
 ↪        ],
        [0.        , 0.13252496, 0.
 ↪55690873]])
```

20.3 Tropický strom

Model je studován ve výukovém materiálu https://www.youtube.com/watch?v=cMQd-okvS_M

```
L = np.matrix([0.3, 0, 0, 100, 150,
               0.05, 0.4, 0, 0, 0,
               0, 0.1, 0.6, 0, 0,
               0, 0, 0.1, 0.7, 0,
               0, 0, 0, 0.1, 0.99]).
 ↪reshape(5,5)
L
```

```
matrix([[  0.3 ,   0.  ,   0.  , 100.
 ↪  , 150.  ],
        [  0.05,   0.4 ,   0.  ,   0.
 ↪  ,   0.  ],
```

(continues on next page)

(pokračujte na předchozí stránce)

```
        [  0.  ,   0.1 ,   0.6 ,   0.
↪  ,   0.  ],
        [  0.  ,   0.  ,   0.1 ,   0.
↪7 ,   0.  ],
        [  0.  ,   0.  ,   0.  ,   0.
↪1 ,   0.99]])
```

```
vlastni_cisla, vlastni_vektory = np.
↪linalg.eig(L)
maximum_index = np.argmax(vlastni_cisla)
maximum_cislo = vlastni_cisla[maximum_
↪index].real

W = vlastni_vektory[:,maximum_index].
↪real
V = (vlastni_vektory**(-1))[maximum_
↪index,:].real

matice_sensitivity = np.around(V.T*W.T,
↪decimals=5)
```

```
matice_elasticity = np.array(matice_
↪sensitivity)*np.array(L)/maximum_cislo
matice_elasticity
```

```
array([[0.02796033, 0.        , 0.
↪     , 0.03259729, 0.04210483],
       [0.07496743, 0.04257206, 0.
↪     , 0.        , 0.        ],
       [0.        , 0.07496743, 0.
↪08918076, 0.        , 0.        ],
       [0.        , 0.        , 0.
↪07496743, 0.12977163, 0.        ],
       [0.        , 0.        , 0.
↪     , 0.04253313, 0.36811714]])
```

```
matice_elasticity.sum(axis=0)
```

```
array([0.10292776, 0.11753949, 0.
↪16414819, 0.20490205, 0.41022197])
```

```
pW = 100*W/np.linalg.norm(W,1)
pW
```

```
matrix([[91.6103505 ],
        [ 6.50284929],
        [ 1.28925986],
        [ 0.31881893],
        [ 0.27872141]])
```

```
sum(pW)
```

```
matrix([[100.]])
```

```
V/V[0,0]
```

```
matrix([[   1.        ,   16.
↪08772469,  113.31971822,  571.
↪56905993,
         1311.34660023]])
```

20.4 Kareta obecná (Loggerhead sea turtle)

Model života Karety obecné je podle publikace Deborah T. Crouse, Larry B. Crowder, Hal Caswell: A Stage-Based Population Model for Loggerhead Sea Turtles and Implications for Conservation, Ecology, Vol. 68, No. 5 (Oct., 1987), pp. 1412-1423.

V tomto modelu autoři studují dříve publikovaný model založený na matici o velikosti 54×54 (Frazer) a místo stáří dělí populaci do vývojových stadií. Tím dosáhnou při zachování přesnosti modelu výrazného snížení řádu matice, což umožňuje detailnější analýzu.

Jendotlivé třídy jsou (podle [15]) (1) vajíčka a vylíhlá mláďata, (2) mladší juvenilní jedinci, (3) starší juvenilní jedinci, (4) subadultní jedinci, (5) poprvé se množící, (6) jednoletí migranti a (7) dospělí. Třídy 5, 6 a 7 jsou uvažovány samostatně kvůli velkým rozdílům ve fertilitě (Frazer 1984). Přibližný věk v jednotlivých kategoriích je pod jeden rok, 1 až 7 let, 8 až 15 let, 16 až 21 let, 22 let, 23 let a 24 až 54 let.

```
np.set_printoptions(suppress=True,␣
↪linewidth=100);
```

```
N = 7
M = np.zeros([N,N])
M[[0],:] = [0,0,0,0,127,4,80]
M[np.arange(N),np.arange(N)] = [0,0.
↪7370,0.6610, 0.6907, 0,0,0.8089]
M[np.arange(1,N),np.arange(N-1)] = [0.
↪6747, 0.0486, 0.0147, 0.0518, 0.8091,␣
↪0.8091]
M = np.matrix(M)
M
```

```
matrix([[  0.    ,   0.    ,   0.    ␣
↪,   0.    , 127.    ,   4.    , ␣
↪80.    ],
        [  0.6747,   0.737 ,   0.    ␣
↪,   0.    ,   0.    ,   0.    , ␣
↪0.    ],
        [  0.    ,   0.0486,   0.661␣
↪,   0.    ,   0.    ,   0.    , ␣
↪0.    ],
```

(continues on next page)

(pokračujte na předchozí stránce)

```
        [  0.    ,   0.    ,   0.
↪0147,   0.6907,   0.    ,   0.    ,
↪   0.    ],
        [  0.    ,   0.    ,   0.    ␣
↪,   0.0518,   0.    ,   0.    ,  ␣
↪0.    ],
        [  0.    ,   0.    ,   0.    ␣
↪,   0.    ,   0.8091,   0.    ,  ␣
↪0.    ],
        [  0.    ,   0.    ,   0.    ␣
↪,   0.    ,   0.    ,   0.8091,  ␣
↪0.8089]])
```

```
vlastni_cisla, vlastni_vektory = np.
 ↪linalg.eig(M)

maximum_index = vlastni_cisla.argmax()
vlastni_cisla[maximum_index]
```

```
(0.945030980691004+0j)
```

```
w = vlastni_vektory[:,[vlastni_cisla.
 ↪argmax()]].real
w = w/sum(w)*100
w
```

```
matrix([[20.65048418],
        [66.97503241],
        [11.4599702 ],
        [ 0.66237138],
        [ 0.03630657],
        [ 0.03108432],
        [ 0.18475094]])
```

```
np.log(vlastni_cisla[maximum_index].
 ↪real)
```

```
-0.056537568225767096
```

Levý vlastní vektor příslušný dominantní vlastní hodnotě *reprodukční hodnota* příslušné třídy. Jedná se o celkový příspěvek třídy a všech pozdějších tříd k populačnímu růstu vyjádřený počtem potomků uvažovaných různou vahou podle toho, zda se rodí do velké či malé populace a vyjádřený v násobcích reprodukční hodnoty nejmladší kategorie. (vektor je normalizovaný tak, aby první komponenta byla rovna jedné).

```
leve_vlastni_vektory = vlastni_
 ↪vektory**(-1)
(leve_vlastni_vektory[maximum_index]/
 ↪leve_vlastni_vektory[maximum_index,
 ↪0]).real
```

```
matrix([[  1.        ,   1.40066842,␣
 ↪  5.99552313, 115.84451127, 568.
 ↪78085247, 507.37304018,
         587.66931373]])
```

```
W = vlastni_vektory[:,maximum_index].
 ↪real
V = (vlastni_vektory**(-1)).
 ↪conjugate()[maximum_index,:].T.real

matice_sensitivity = np.around(V*W.T,
 ↪decimals=5)
matice_elasticity = np.array(matice_
 ↪sensitivity)*np.array(M)/vlastni_
 ↪cisla[maximum_index]
matice_elasticity = matice_elasticity.
 ↪real
matice_elasticity
```

```
array([[0.        , 0.        , 0.  ␣
 ↪     , 0.        , 0.01209484, 0.
 ↪00033861, 0.03894052],
       [0.05100422, 0.18068776, 0.  ␣
 ↪     , 0.        , 0.        , 0. ␣
 ↪      , 0.        ],
       [0.        , 0.0510021 , 0.
 ↪11868933, 0.        , 0.        ,␣
 ↪0.        , 0.        ],
       [0.        , 0.        , 0.
 ↪05100204, 0.13850822, 0.        ,␣
 ↪0.        , 0.        ],
       [0.        , 0.        , 0.  ␣
 ↪     , 0.05100187, 0.        , 0. ␣
 ↪      , 0.        ],
       [0.        , 0.        , 0.  ␣
 ↪     , 0.        , 0.03895539, 0. ␣
 ↪      , 0.        ],
       [0.        , 0.        , 0.  ␣
 ↪     , 0.        , 0.        , 0.
 ↪03863005, 0.2295232 ]])
```

```
matice_elasticity.sum(axis=0)*100
```

```
array([ 5.100422  , 23.16898583, 16.
 ↪96913649, 18.95100866,  5.10502311,
 ↪  3.89686611, 26.84637225])
```

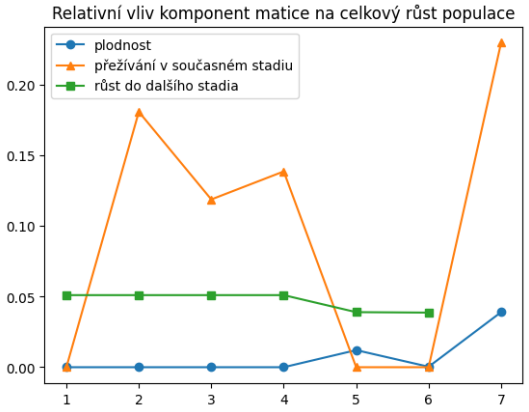
V původní publikaci, kde se objevil tento model, je vliv jednotlivých komponent matice vyjádřen i graficky. Pokusíme se tento obrázek zreprodukovat.

```
stav = np.arange(1,N+1)
plt.plot(stav, matice_elasticity[0,:],
 ↪"o-",label = "plodnost")
plt.plot(stav, matice_elasticity[np.
 ↪arange(N),np.arange(N)], "^-", label=
 ↪"přežívání v~současném stadiu")
```

(continues on next page)

(pokračujte na předchozí stránce)

```
plt.plot(stav[:-1], matice_
 ↪elasticity[np.arange(1,N),np.arange(N-
 ↪1)], "s-", label="růst do dalšího␣
 ↪stadia")

plt.legend()
plt.title("Relativní vliv komponent␣
 ↪matice na celkový růst populace");
```



Výstupem z modelu je skutečnost, že při snaze zachránit populaci karety má ochrana vajíček, na kterou se tradičně zaměřovala pozornost, jenom malý vliv na celkovou kondici populace. Účelnější je zaměřit se na přežívání dospívajících a dospělých želv. Toho je možné dosáhnout úpravou rybářských sítí tak, aby z nich želvy mohly uniknout. Viz Turtle Excluder Device.

KAPITOLA 21

Levinsův model metapopulací

Americký ekolog Richard Levins představil v roce 1969 model metapopulací. Jedná se o populace žijící ve fragmentovaném prostředí, které je možno studovat jako systém oddělených populací, které spolu komunikují díky možnosti migrace. Nejčastěji dochází k fragmentaci vlivem činnosti člověka. V hustě osídlených oblastech, jako je například Evropa, je fragmentace důležitým aspektem, ovlivňujícím vývoj životního prostředí. Proto je vhodné podrobně rozumět jeho souvislostem a uvědomovat si důsledky fragmentace i to, jak činnost člověka vývoj metapopulací ovlivňuje.

21.1 Populace ve fragmentovaném prostředí

Uvažujme populaci žijící ve fragmentovaném prostředí. Tuto fragmentaci si můžeme představit jako systém ostrůvků, ve kterých se populace nachází. Tyto ostrůvky (fragmenty) jsou zřetelně odděleny. (Například louky nebo lesní palouky.)

Některé z těchto podlokalit mohou být nevhodné pro trvalé osídlení a populace na nich může přežívat jen díky přistěhovalcům.

Na některých podlokalitách zase může vlivem náhodných jevů populace zcela vymřít a za několik let se zde může znovu objevit, díky přenesení několika jedinců z jiné podlokality, kteří se zde rozmnoží.

Předpokládejme, že míra migrace je tak malá, že není možné posuzovat populaci jako jedinou populaci, která se řídí logistickou nebo jinou obdobnou rovnicí. Taková síť místních malých populací, které jsou propojeny občasnými náhodnými migracemi, se nazývá *metapopulace*.

21.2 Matematický model

Budeme sledovat, jaká část podlokalit (fragmentů) bude danou populací obsazena.

Označme $n(t)$ počet fragmentů, které jsou v čase t obsazeny sledovanou populací, N celkový počet fragmentů, $d(n)$ rychlost vymírání lokálních populací, tj. počet lokálních populací které vyhynou za jednotku času v případě, že je obsazeno právě n fragmentů a konečně označme $b(n)$ rychlost kolonizace nových fragmentů, tj. počet fragmentů prostředí, které jsou za jednotku času nově kolonizovány za předpokladu, že populace je rozdělena do n fragmentů životního prostředí.

Relativní četnost obsazených fragmentů v čase t označme $x(t)$, platí tedy $x(t) = \frac{n(t)}{N}$. Tato četnost se bude (podle předpokladů) řídit diferenciální rovnicí

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \frac{1}{N}\frac{\mathrm{d}n}{\mathrm{d}t} = \frac{b(n) - d(n)}{N}.$$

Funkce b a d musí splňovat přirozené podmínky $b(0) = b(N) = 0$, $b(n) \geq 0$ pro $n \geq 0$, $d(0) = 0$, $d(n) \geq 0$. Tyto podmínky vyjadřují, že pokud všechny fragmenty jsou prázdné nebo obsazené, nemůže dojít ke kolonizaci, protože buď nejsou žádní potenciální kolonizátoři, nebo není co kolonizovat a pokud není žádný fragment kolonizován, žádná populace nevymírá. Jednoduché funkce, splňující tyto podmínky, jsou

$$b(n) = an(N - n), \quad d(n) = \nu n.$$

Po dosazení do rovnice

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \frac{an(N-n) - \nu n}{N} = aNx(1-x) - \nu x$$

a po zavedení konstanty $\mu = aN$, obdržíme matematický model

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \mu x(1-x) - \nu x.$$

Jediné realistické hodnoty veličiny x jsou z intervalu $[0, 1]$. Veličina x navíc již je v bezrozměrných jednotkách.

21.3 Důsledky modelu

Často citované závěry z modelu jsou následující (viz [9]).

- I když na jednotlivých fragmentech životního prostředí může populace vymizet, díky občasné migraci může populace jako celek trvale přežívat.
- Za podmínek vhodných k přežívání populace se poměr obsazených a neobsazených fragmentů životního prostředí ustálí na jisté konstantní hodnotě, dané stabilním stacionárním stavem. Z tohoto důvodu snížení počtu neobsazených fragmentů vede ve svém důsledku k odpovídajícímu snížení počtu fragmentů obsazených. Proto pro zachování optimálních podmínek pro vývoj populace je nutno chránit nejen fragmenty obsazené, ale i neobsazené, které slouží jako jistá záloha. Do nich se může populace přenést při vyhynutí na jiných fragmentech. Zničení volných fragmentů vede k tomu, že tyto zálohy nejsou k dispozici a v případě vyhynutí populace na některém z fragmentů se život nemá kam přenést a opět se nastolí rovnovážný poměr mezi počtem fragmentů obsazených a neobsazených. Přitom zničením fragmentu v tomto smyslu rozumíme jakékoliv zabránění populaci, aby se díky migraci mohla na tento fragment přesunout. Nemusí se tedy jednat pouze o fyzickou destrukci dané lokality, ale i například o uzavření přirozeného koridoru, který migraci umožňuje a kterým může být daná lokalita kolonizována.
- Ve fragmentovaném prostředí podle výše uvedeného modelu zůstávají vždy nějaké neobsazené lokality. To může být výhodou pro jiné, konkurenčně slabší biologické druhy, které nemůžou s naším uvažovaným druhem koexistovat, protože při vzájemné koexistenci podlehnou konkurenčnímu boji a vyhynou (viz některé modely dále). Vzhledem k tomu, že některé fragmenty jsou neobsazeny, stačí, aby konkurenčně slabší druh byl schopný rychlejší migrace než konkurenčně silnější druh, a může dojít k tomu, že oba druhy budou koexistovat (nikoliv však na téže lokalitě).

Levinsovu modelu je často vyčítáno přílišné zjednodušení (nerozlišování lokálních populací, konstantní parametry extinkce a kolonizace, ignorování prostorového rozložení fragmentů a jiné). Argumenty pro přijetí tohoto modelu však tyto námitky zpravidla vyvrací. Od poukázání na fakt, že kvůli zjednodušením modely vlastně děláme, až po sofistikovaný rozbor toho, že se model dá snadno při mírně modifikované interpretaci proměnných použít i v mnohem obecnějším pojetí. Viz například [7].

Klasickými příklady metapopulací jsou (podle [15]) lesní hmyz žijící na kmenech padlých stromů a hmyz obecně, dafnie ve skalních jezírkách, skokani v rybnících, brhlíci v remízcích obklopených zemědělskou krajinou.

21.4 Možné úkoly

1. Prozkoumejte dynamiku modelu. Nakreslete řešení pro různé hodnoty parametrů. Snažte se počet parametrů zredukovat zavedením bezrozměrného času (velikost populace již bezrozměrná je).
2. Pro stejné hodnoty bezrozměrného parametru získaného v předchozím případě máme stejné chování (konvergenci do stejného stavu), ale liší se rychlost konvergence k tomuto stavu. Nakreslete pro rovnici

   $$\frac{\mathrm{d}x}{\mathrm{d}t} = \mu x(1-x) - \nu x$$

   vývoj řešení v čase pro různé kombinace parametrů. Volte hodnoty tak, aby byla patrná konvergence do stejného stavu, ale jiná rychlost této konvergence.
3. Jedno navrhované zobecnění Levinsova modelu je v [9] ve tvaru

   $$\frac{\mathrm{d}x}{\mathrm{d}t} = \mu x^{\alpha}(1-x)^{\beta} - \nu x^{\gamma}.$$

   Prozkoumejte tento model z hlediska závislosti na parametrech. Jeden z efektů zavedení parametrů α a γ je jiné chování v okolí nuly v závislosti na vzájemné velikosti těchto parametrů. Pokuste se zjistit jak se liší chování pro $\alpha > \gamma$ a $\alpha < \gamma$ prozkoumáním růstových křivek. Potvrďte si hypotézu prozkoumáním řešení modelu při uvedeném nastavení parametrů.
4. Další navrhované zobecnění je v [15] str. 345 ve tvaru

   $$\frac{\mathrm{d}x}{\mathrm{d}t} = \mu x(1-x) - \nu_0 e^{-mx} x.$$

   Porovnejte chování tohoto modelu a modelu původního, který je zde obsažen pro volbu $m = 0$.
5. Další možné modifikace Levinsova modelu jsou popsány na webech EcoVirtual a Primer of Ecology using R a v článku Gotelli, N J. 1991. *Metapopulation Models: The Rescue Effect, the Propagule Rain, and the Core-Satellite Hypothesis.* The American Naturalist 138: 768–76 (k dispozici u vyučujícího).

KAPITOLA 22

Modely konkurence populací

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.integrate import solve_ivp
```

22.1 Lotkův–Volterrův model konkurence dvou populací

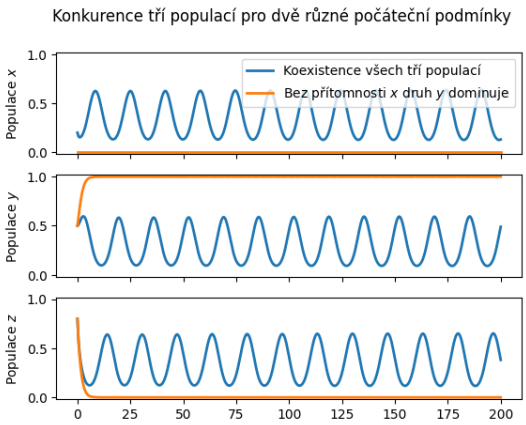
22.2 Konkurence tří populací

```
meze = [0,200]
def konkurence_tri(t,X,a=1,b=1,c=0.4,
 ↪d=1.4,alpha=1,beta=1.5,gamma=1,
 ↪delta=0.5,m=1,n=0.7,o=1.2,p=1):
    x,y,z = X
    return [(a-b*x-c*y-d*z)*x, (alpha-
 ↪beta*x-gamma*y-delta*z)*y,(m-n*x-o*y-
 ↪p*z)*z]
parametry = (1, 1,   0.3, 1.6,
             1, 1.8, 1,   0.2,
             1, 0.3, 1.8, 1  )

pocatecni_podminky = [ [.2,.5,0.8],
                       [0,.5,0.8],
                     ]

t = np.linspace(*meze,400)
reseni = [
    solve_ivp(
        konkurence_tri,
        meze,
        pocatecni_podminka,
        t_eval=t,
        args=parametry
        )
    for pocatecni_podminka in pocatecni_
 ↪podminky];
popisky=[rf"Populace ${i}$" for i in ["x
```

(continues on next page)

(pokračujte na předchozí stránce)

```
 ↪","y","z"]]
fig,ax = plt.subplots(3,1,sharex=True)

for barva,res in enumerate(reseni):
    for i in range(len(ax)):
        ax[i].plot(t,res.y[i],color="C
 ↪"+str(barva),lw=2)
        ax[i].set(
            ylim=(-0.02,1.02),
            ylabel=popisky[i],
        )
plt.suptitle("Konkurence tří populací␣
 ↪pro dvě různé počáteční podmínky")
ax[0].legend(["Koexistence všech tří␣
 ↪populací",r"Bez přítomnosti $x$ druh
 ↪$y$ dominuje"]);
```



22.2.1 Úkol

1. Zaujalo vás oscilatorické chování systému? Chová se takto systém skutečně, nebo jde jenom o nějakou zaokrouhlovací a numerickou chybu. Navrhněte postup, jak toto zjistit. Návod: pročtěte si parametry příkazu `solve_ivp` a promyslete, který má vliv na přesnost aproximace řešení.
2. Zkuste v systému nasadit ještě silnější míru konkurence. Všechny koeficienty mezidruhové konkurence, které jsou větší než jedna, zkuste dále navyšovat a sledovat, jestli neuvidíte nějaké další zajímavé chování.

KAPITOLA 23

Modely dravce a kořisti

23.1 Lotkův a Volterrův mode dravce a kořisti

23.2 Model dravce a kořisti s Hollingovou trofickou funkcí

23.3 Model dravce a dvou populací kořisti

23.4 Model dravce a kořisti s kanibalismem dravce

KAPITOLA 24

Práce na zápočtovém projektu

Práce na zápočtovém projektu nebo výběr vhodného tématu.

Část III

Python konstrukce a ukázky

KAPITOLA 25

Základní konstrukce

```
import numpy as np
```

25.1 Sestavení seznamu

```
# Sestaveni posloupnosti. Prvních pět␣
 ↪celých čísel počínaje nulou
# se použije k~sestavení seznamu␣
 ↪druhých mocnin. Ten se uloží do
# nové proměnné.
mocniny = [i**2 for i in range(5)]
```

```
# Vynásobení každého čísla seznamu␣
 ↪dvojkou a uložení nového
# seznamu do nové proměnné.
dvojnasobky_mocnin = [2*i for i in␣
 ↪mocniny]
```

```
# Sestavení posloupnosti typu numpy.
 ↪array. Na tuto posloupnost
# můžeme přímo aplikovat matematické␣
 ↪operace.
seznam = np.array(range(5))
dvojnasobky_mocnin = 2*seznam**2
```

25.2 Cyklus

```
# Cyklus. Zadaným počtem opakování se␣
 ↪provádí odsazený blok.
# Iterační index i nabývá hodnoty 0 až␣
 ↪4.
for i in range(5):
    pruchod = i+1
    hlaseni = f"{pruchod}. Toto se␣
 ↪vytiskne petkrat."
```

(continues on next page)

(pokračujte na předchozí stránce)

```
    print(hlaseni)
print("Uff, hotovo.")
```

```
1. Toto se vytiskne petkrat.
2. Toto se vytiskne petkrat.
3. Toto se vytiskne petkrat.
4. Toto se vytiskne petkrat.
5. Toto se vytiskne petkrat.
Uff, hotovo.
```

```
# Iterace přes dvojici seznamů současně
druh = ["pes","kapr","roháč"]
skupina = ["savec", "vánoční jídlo",
 ↪"brouk"]
for i,j in zip(druh,skupina):
    print(f"{i} je {j}")
```

```
pes je savec
kapr je vánoční jídlo
roháč je brouk
```

```
# Iterace přes dvojici seznamů současně.
 ↪ Prvním seznamem jsou indexy
# prvků v~druhém seznamu (pořadí␣
 ↪počítáno od nuly).
druh = ["pes","kapr","roháč"]
for i,j in enumerate(druh):
    print(f"{j} je na pozici {i}")
```

```
pes je na pozici 0
kapr je na pozici 1
roháč je na pozici 2
```

25.3 Větvení

```
# Větvení. Podle podmínky se provede␣
 ↪přísušný odsazený blok.
a = 2
if a<4 :
    print("Malá hodnota v~proměnné a.")
    print("Podmínka je splněna.")
else:
    print("Velká hodnota v~proměné a.")
    print("Podmínka neni splněna.")
```

```
Malá hodnota v~proměnné a.
Podmínka je splněna.
```

KAPITOLA 26

Cykly

Při opakování kódu většinou známe počet opakování a používáme cyklus `for`. Za deklarací použité proměnné pro řízení cyklu a za seznamem, přes který proměnná iteruje, je dvojtečka. Blok příkazů, které se opakují, je odsazen o čtyři mezery.

Níže jsou základní techniky.

26.1 Prostá iterace

```
for i in range(5):
    print("Toto se tiskne pětkrát")
```

```
Toto se tiskne pětkrát
Toto se tiskne pětkrát
Toto se tiskne pětkrát
Toto se tiskne pětkrát
Toto se tiskne pětkrát
```

26.2 Iterace přes seznam

Často iterujeme přes seznam hodnot.

```
stromy = ["buk", "smrk", "jedle", "dub"]
for strom in stromy:
    print(f"Můj oblíbený strom je
 ↪{strom}.")
```

```
Můj oblíbený strom je buk.
Můj oblíbený strom je smrk.
Můj oblíbený strom je jedle.
Můj oblíbený strom je dub.
```

V těle cyklu může být sada příkazů na libovolný počet řádků. Pokud je v těle cyklu jeden příkaz a výstup chceme uložit do seznamu, je možné použít i následující variantu.

```
stromy = ["buk", "smrk", "jedle", "dub"]
vysledek = [f"Můj oblíbený strom je
 ↪{strom}." for strom in stromy]
vysledek
```

```
['Můj oblíbený strom je buk.',
 'Můj oblíbený strom je smrk.',
 'Můj oblíbený strom je jedle.',
 'Můj oblíbený strom je dub.']
```

26.3 Iterace přes více seznamů současně

Pomocí `zip` je možné iterovat přes více seznamů současně.

```
stromy = ["buk", "smrk", "tis", "dub"]
lide = ["Jan", "Jana", "Tom", "Jiří"]
for clovek,strom in zip(lide,stromy):
    print(f"{clovek} má na zahrádce
 ↪{strom}.")
```

```
Jan má na zahrádce buk.
Jana má na zahrádce smrk.
Tom má na zahrádce tis.
Jiří má na zahrádce dub.
```

26.4 Iterace se sledováním pozice v seznamu

Někdy je vhodné při iterování znát i pozici v seznamu, která je právě zpracovávána. Toho dosáhneme pomocí `enumerate`.

```
stromy = ["buk", "smrk", "tis", "dub"]
lide = ["Jan", "Mirek", "Tom", "Jiří"]
for i,strom in enumerate(stromy):
    print(f"{lide[i]} má na zahrádce
 ↪{strom}.")
```

```
Jan má na zahrádce buk.
Mirek má na zahrádce smrk.
Tom má na zahrádce tis.
Jiří má na zahrádce dub.
```

26.5 Praktická poznámka

Zpravidla nejprve odladíme jednu otočku cyklu a potom doplníme volání cyklu a odsazení těla cyklu.

```
clovek = "Adam"
strom = "třešeň"
strom = strom.capitalize()
print(f"{strom} zasadil {clovek}.")
```

```
Třešeň zasadil Adam.
```

```
stromy = ["buk", "smrk", "tis", "dub"]
lide = ["Jan", "Mirek", "Tom", "Jiří"]
for clovek,strom in zip(lide,stromy):
    strom = strom.capitalize()
    print(f"{strom} zasadil {clovek}.")
```

```
Buk zasadil Jan.
Smrk zasadil Mirek.
Tis zasadil Tom.
Dub zasadil Jiří.
```

26.6 Předčasné ukončení

Cyklus je možno opustit i předčasně pomocí `break`. Použití má smysl uvnitř nějaké podmínky. Uvnitř podmínky je další odsazení.

Příkaz `break` ukončí celý cyklus. Je možné použít i příkaz `continue`, který přeruší pouze danou otočku cyklu a přejde k další otočce v pořadí.

```
for i in range(10):
    druha_mocnina = i**2
    if druha_mocnina > 30:
        print("Druhá mocnina dosáhla␣
 ↪vysokou hodnotu, končím.")
        # Pokud je druha mocnina cisla␣
 ↪i vetsi nez 30, prerusime cyklus
        break
    print (f"Druhá mocnina čísla {i} je
 ↪{druha_mocnina}.")
# Kontrola, kdy se cyklus zastavil     ␣
 ↪
i
```

```
Druhá mocnina čísla 0 je 0.
Druhá mocnina čísla 1 je 1.
Druhá mocnina čísla 2 je 4.
Druhá mocnina čísla 3 je 9.
Druhá mocnina čísla 4 je 16.
Druhá mocnina čísla 5 je 25.
Druhá mocnina dosáhla vysokou␣
 ↪hodnotu, končím.
```

```
6
```

26.7 Praktická ukázka

Vyřešíme diferenciální rovnici logistického růstu pro různé hodnoty nosné kapacity. Data schováme v tabulce a nakonec je vykreslíme. Jednotlivé otočky cyklu jsou použity k vytvoření sloupců s daty.

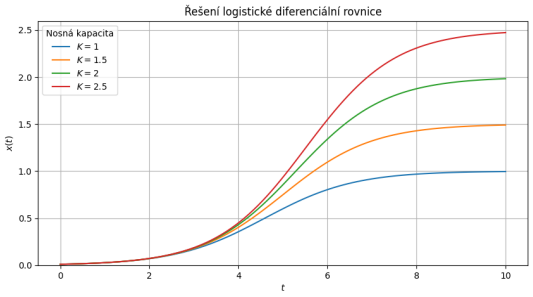
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.integrate import solve_ivp
```

```
pocatecni_podminka = [0.01]
meze = [0,10]
pocet_hodnot = 100
hodnoty_K = [1,1.5,2,2.5]

def rovnice(t, x, r=1, K=1):
    return r*x*(1-x/K)

df = pd.DataFrame()
df["t"] = np.linspace(*meze,pocet_
 ↪hodnot)

for K in hodnoty_K:
    reseni = solve_ivp(
                    rovnice,
                    meze,
                    pocatecni_podminka,
                    t_eval=df["t"],
```

(continues on next page)

(pokračujte na předchozí stránce)

```
                    args=(1,K)
                    ).y[0]
    df[K] = reseni

fig,ax = plt.subplots(figsize=(10,5))

df.plot(x="t", ax=ax)

ax.set(
    ylim = (0,None),
    title = "Řešení logistické␣
 ↪diferenciální rovnice",
    xlabel=r"$t$",
    ylabel=r"$x(t)$",
)
ax.legend([f"$K={K}$" for K in hodnoty_
 ↪K], title="Nosná kapacita")
ax.grid()
```



KAPITOLA 27

Kreslení funkcí jednoduše

Jednoduché úkoly je vhodné řešit jednoduše. Například pokud chceme vykreslit několik funkcí do jednoho grafu, prostě postupně voláme příkazy kreslící jednotlivé funkce.

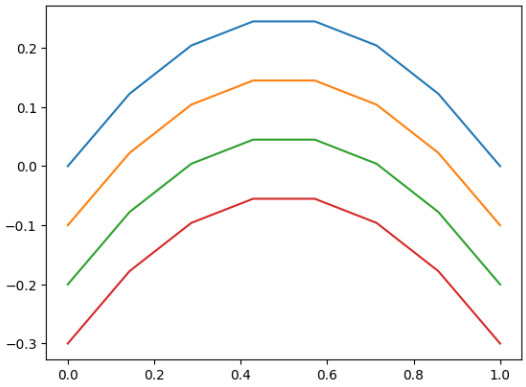
Je dobré nejprve zavolat knihovny a nastavit proměnné, poté vykrelist grafy a poté dodělat kosmetické úpravy. To je v jednotlivých políčkách. Na konci práce bude asi vhodné poslední čtyři políčka spojit. Zkontrolujte, že jste v příkazovém modu (needitujete políčko, nalevo je modrá čára a ne zelená), označte spojované buňky (shift plus šipky) a spojte (shift + M). Případně neznačujte a shift+M připojí následující buňku. Stejnou práci odvede Menu -> Edit -> Merge Cell Bellow).

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

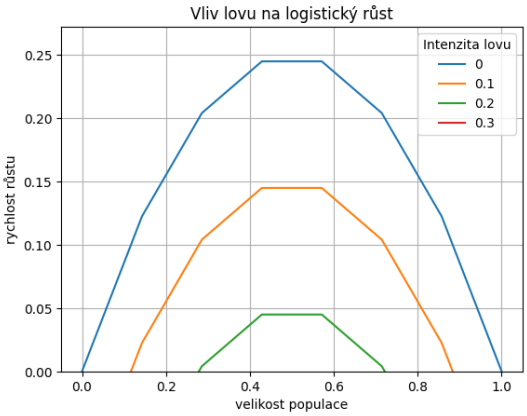
```
N = 8
meze = [0,1]
lov = [0,0.1,0.2,0.3]
```

```
x = np.linspace(*meze,N)
def logisticky_rust(x,h=0):
    """
    Funkce definuje pravou stranu␣
 ↪logisticke rovnice s~pridanym lovem␣
 ↪konstantni intenzity.
    """
    r = 1
    K = 1
    return(x*r*(1-x/K) - h)
```

```
fig, ax = plt.subplots()
for h in lov:
    plt.plot(x,logisticky_rust(x,h=h))
```




```
ax.legend(lov, title="Intenzita lovu")
ax.set(title="Vliv lovu na logistický␣
 ↪růst",
       xlabel="velikost populace",
       ylabel="rychlost růstu",
       ylim=(0,None))
ax.grid()
fig
```



KAPITOLA 28

Kreslení funkcí pro pokročilé

Jednoduché úkoly je vhodné řešit jednoduše a nejjednodušší je data kreslit do grafu rovnou po výpočtu. Ve složitějších modelech se však data počítají dlouho a je proto nejlepší je nejprve vypočítat a potom vykreslit. K ukládání se hodí tabulky. Snadné je použití knihovny Pandas.

Je dobré nejprve zavolat knihovny a nastavit proměnné, sestavit tabulku, vykreslit grafy a poté dodělat kosmetické úpravy. To je v jednotlivých políčkách. Na konci práce bude asi vhodné poslední čtyři políčka spojit. Zkontrolujte, že jste v příkazovém modu (needitujete políčko, nalevo je modrá čára a ne zelená), označte spojované buňky (shift plus šipky) a spojte (shift + M). Případně neznačujte a shift+M připojí následující buňku. Stejnou práci odvede Menu -> Edit -> Merge Cell Bellow).

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
N = 500
meze = [0,1]
lov = [0,0.1,0.2,0.3]
```

```
x = np.linspace(*meze,N)
def logisticky_rust(x,h=0):
    """
    Funkce definuje pravou stranu␣
 ↪logisticke rovnice s~pridanym lovem␣
 ↪konstantni intenzity.
    """
    r = 1
    K = 1
    return(x*r*(1-x/K) - h)
```

```
# První metoda, výpočet dat a poté␣
 ↪vložení do tabulky
data = [logisticky_rust(x,h=h) for h in␣
 ↪lov]
df = pd.DataFrame(data).T
```

(continues on next page)

(pokračujte na předchozí stránce)

```
df.columns = lov
df["x"] = x

# Druhá metoda, skládání tabulky po␣
 ↪sloupcích
# df = pd.DataFrame()
# df["x"] = x
# for h in lov:
#     df[h] = logisticky_rust(x,h=h)

# Varianta obou, data vypočteme po␣
 ↪sloupcích a potom sestavíme tabulku
#
# data = np.full([len(x),len(lov)],np.
 ↪nan)
# for i,h in enumerate(lov):
#     data[:,i] = logisticky_rust(x,h=h)
# df = pd.DataFrame(data, columns=lov)
# df["x"] = x
df
```

```
          0.0       0.1       0.2   ␣
 ↪   0.3         x
0    0.000000 -0.100000 -0.200000 -0.
 ↪300000  0.000000
1    0.002000 -0.098000 -0.198000 -0.
 ↪298000  0.002004
2    0.003992 -0.096008 -0.196008 -0.
 ↪296008  0.004008
3    0.005976 -0.094024 -0.194024 -0.
 ↪294024  0.006012
4    0.007952 -0.092048 -0.192048 -0.
 ↪292048  0.008016
..        ...       ...       ...   ␣
 ↪   ...       ...
495  0.007952 -0.092048 -0.192048 -0.
 ↪292048  0.991984
496  0.005976 -0.094024 -0.194024 -0.
 ↪294024  0.993988
497  0.003992 -0.096008 -0.196008 -0.
 ↪296008  0.995992
```

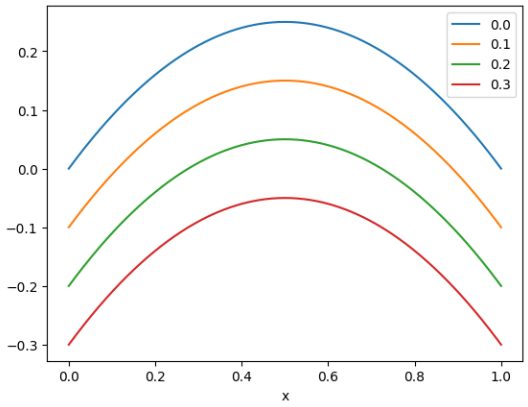
(continues on next page)

(pokračujte na předchozí stránce)

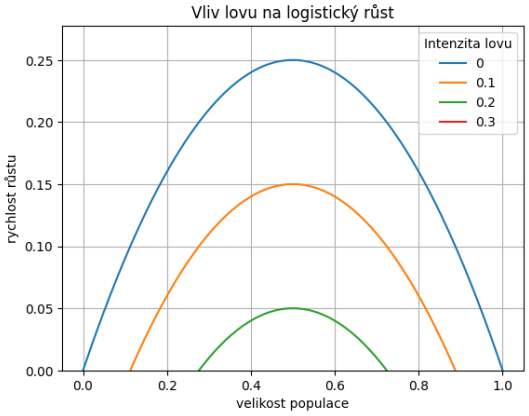
```
498  0.002000 -0.098000 -0.198000 -0.
↪298000  0.997996
499  0.000000 -0.100000 -0.200000 -0.
↪300000  1.000000

[500 rows x 5 columns]
```

```
fig, ax = plt.subplots()
df.plot(x="x", ax=ax)
```

```
<Axes: xlabel='x'>
```




```
ax.legend(lov, title="Intenzita lovu")
ax.set(title="Vliv lovu na logistický␣
↪růst",
       xlabel="velikost populace",
       ylabel="rychlost růstu",
       ylim=(0,None))
ax.grid()
fig
```



KAPITOLA 29

Diferenciální rovnice pro začátečníky

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.integrate import solve_ivp
```

```
pocatecni_podminka = [0.1]
meze = [0,10]

def rovnice(t, x, h=0):
    r = 1
    K = 1
    return r*x*(1-x/K) - h
```

```
reseni = solve_ivp(
                   rovnice,
                   meze,
                   pocatecni_podminka,
                   dense_output=True,
                   # t_eval=np.
↪linspace(*meze,500), ## zde je možné␣
↪zadat pole hodnot, kde se mají určit␣
↪řešení
                   # max_step=np.Inf, #
↪# defaultní je krok libovolné délky
                   )
reseni
```

```
  message: The solver successfully␣
↪reached the end of the integration␣
↪interval.
  success: True
   status: 0
        t: [ 0.000e+00  1.023e-01  1.
↪126e+00  2.672e+00  4.488e+00
             5.352e+00  6.217e+00  7.
↪970e+00  9.831e+00  1.000e+01]
        y: [[ 1.000e-01  1.096e-01 ␣
↪2.551e-01  6.167e-01  9.082e-01
              9.592e-01  9.824e-01 ␣
↪9.967e-01  9.994e-01  9.995e-01]]
```

(continues on next page)

(pokračujte na předchozí stránce)

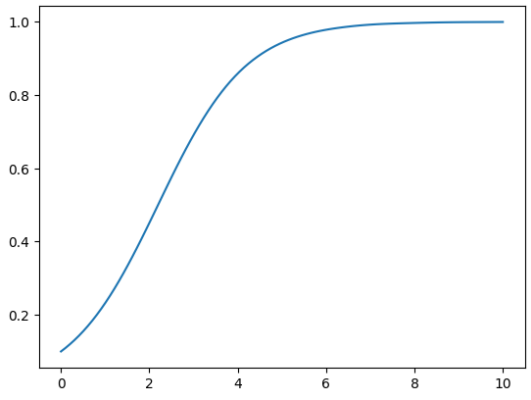
```
      sol: <scipy.integrate._ivp.
↪common.OdeSolution object at␣
↪0x7f176e2f4ed0>
 t_events: None
 y_events: None
     nfev: 62
     njev: 0
      nlu: 0
```

```
print(reseni.sol(2.3))  # reseni v~case␣
↪2.3
```

```
[0.52575795]
```

```
t=np.linspace(*meze, 100)  # graf reseni
fig,ax = plt.subplots(1)
ax.plot(t,reseni.sol(t).T)
```

```
[<matplotlib.lines.Line2D at␣
↪0x7f17ac5dd010>]
```

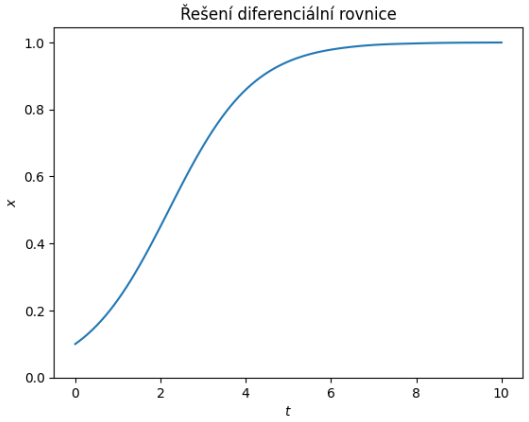



```
ax.set(
    ylim = (0,None),
```

(continues on next page)

(pokračujte na předchozí stránce)

```
    title = "Řešení diferenciální␣
 ↪rovnice",
    xlabel=r"$t$",
    ylabel=r"$x$",
)
fig
```




```
df = pd.DataFrame() #tabulka␣
 ↪s~hodnotami řešení ve více bodech
moje_t = np.array([3,5,7])
df["t"] = moje_t
df["x"] = reseni.sol(moje_t).T
df
```

	t	x
0	3	0.690739
1	5	0.942911
2	7	0.992202

KAPITOLA 30

Diferenciální rovnice pro pokročilejší

Pokročilejší dovedností je například odhalení, jak se rovnice chová při změně parametru.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.integrate import solve_ivp
```

```
pocatecni_podminka = [0.1]
meze = [0,10]
lovy = [0,0.2,0.4]

def rovnice(t, x, h=0):
    """
    Logistický růst a lov konstantním␣
 ↪úsilím, tj za jednotku času se loví␣
 ↪stále stejné procento populace.
    """
    r = 1
    K = 1
    return r*x*(1-x/K) - h*x
```

```
reseni = [solve_ivp(
                rovnice,
                meze,
                pocatecni_podminka,
                dense_output=True,
                args = [lov]
                   ).sol
          for lov in lovy ]
reseni
```

```
[<scipy.integrate._ivp.common.
 ↪OdeSolution at 0x7f077e120bd0>,
 <scipy.integrate._ivp.common.
 ↪OdeSolution at 0x7f077d0cc050>,
 <scipy.integrate._ivp.common.
 ↪OdeSolution at 0x7f073fac3d10>]
```

```
t=np.linspace(*meze, 100)  # graf reseni
```

(continues on next page)

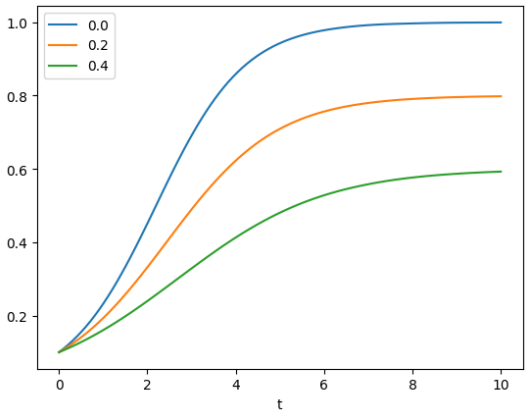
(pokračujte na předchozí stránce)

```
df = pd.DataFrame([r(t)[0] for r in␣
 ↪reseni]).T
df.columns = lovy
df["t"] = t
df
```

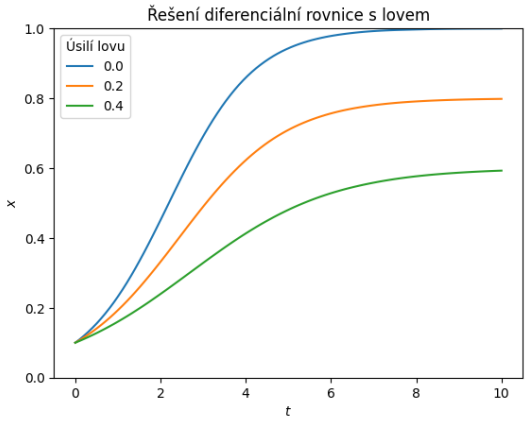
```
          0.0       0.2       0.4     ␣
 ↪    t
0   0.100000  0.100000  0.100000   0.
 ↪00000
1   0.109465  0.107288  0.105153   0.
 ↪10101
2   0.119702  0.115018  0.110513   0.
 ↪20202
3   0.130753  0.123204  0.116081   0.
 ↪30303
4   0.142664  0.131863  0.121861   0.
 ↪40404
..       ...       ...       ...     ␣
 ↪   ...
95  0.999297  0.797415  0.590703   9.
 ↪59596
96  0.999345  0.797608  0.591242   9.
 ↪69697
97  0.999398  0.797787  0.591750   9.
 ↪79798
98  0.999455  0.797953  0.592229   9.
 ↪89899
99  0.999507  0.798110  0.592680  10.
 ↪00000

[100 rows x 4 columns]
```

```
fig,ax = plt.subplots(1)
df.plot(ax=ax, x="t")
```

```
<Axes: xlabel='t'>
```




```
ax.set(
    ylim = (0,1),
    title = "Řešení diferenciální␣
 ↪rovnice s~lovem",
    xlabel=r"$t$",
    ylabel=r"$x$",
)
ax.legend(title="Úsilí lovu")
fig
```



KAPITOLA 31

Tabulky

S tabulkami se pracuje pomocí knihovny Pandas. Narozdíl od Excelu při práci data v tabulce nevidíme, dokud si je nenecháme vypsat. To ale nevadí. A při rozsáhlých tabulkách to je stejně jedno.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

31.1 Tabulky z externího souboru

Častým zdrojem dat je externí soubor. Je možné importovat sešity Excelu, ale nejrychlejší a nejphodlnější je pracovat s csv soubory, což jsou textové soubory s daty oddělenými čárkami. Mohou být na lokálním disku nebo je možné je stáhnout z URL. Po stažení je vhodné si prohlédnout, co máme za data.

```
zdroj = "https://raw.githubusercontent.
↪com/robert-marik/dmp/main/data/hudson_
↪bay_lynx_hare.csv"
df = pd.read_csv(
    zdroj,
    skiprows = 2
)
df
```

```
    Year  Lynx  Hare
0   1900   4.0  30.0
1   1901   6.1  47.2
2   1902   9.8  70.2
3   1903  35.2  77.4
4   1904  59.4  36.3
5   1905  41.7  20.6
6   1906  19.0  18.1
7   1907  13.0  21.4
8   1908   8.3  22.0
```

(continues on next page)

(pokračujte na předchozí stránce)

```
9   1909   9.1  25.4
10  1910   7.4  27.1
11  1911   8.0  40.3
12  1912  12.3  57.0
13  1913  19.5  76.6
14  1914  45.7  52.3
15  1915  51.1  19.5
16  1916  29.7  11.2
17  1917  15.8   7.6
18  1918   9.7  14.6
19  1919  10.1  16.2
20  1920   8.6  24.7
```

Používáme náhled na začátek souboru, konec souboru, nebo náhodný vzorek. Málokdy nás zajímá celá tabulka, protože může mít tisíce řádků.

```
df.head() # podobne tail a sample pro␣
↪konec a nahodny vyber
```

```
   Year  Lynx  Hare
0  1900   4.0  30.0
1  1901   6.1  47.2
2  1902   9.8  70.2
3  1903  35.2  77.4
4  1904  59.4  36.3
```

Často nás zajímají základní statistiky pro sloupce tabulky, rozměry tabulky a zda tabulka obdahuje nedefinované hodnoty.

```
df.describe()
```

```
              Year        Lynx       ␣
↪Hare
count    21.000000  21.000000  21.
↪000000
mean   1910.000000  20.166667  34.
↪080952
```

(continues on next page)

(pokračujte na předchozí stránce)

```
std       6.204837  16.656000  21.
↪413982
min    1900.000000   4.000000   7.
↪600000
25%    1905.000000   8.600000  19.
↪500000
50%    1910.000000  12.300000  25.
↪400000
75%    1915.000000  29.700000  47.
↪200000
max    1920.000000  59.400000  77.
↪400000
```

```
# počet řádků a sloupců
df.shape
```

```
(21, 3)
```

```
# True, pokud nějaká hodnota v~tabulce␣
↪chybí
df.isnull().values.any()
```

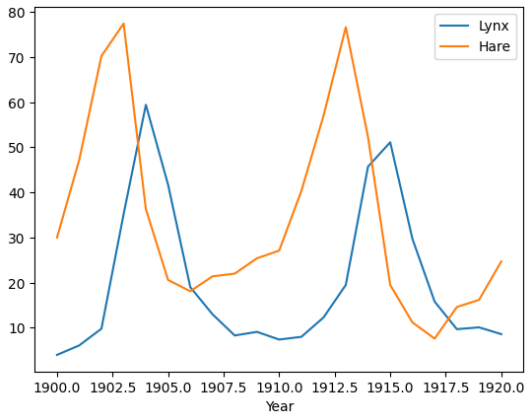
```
False
```

31.2 Grafy

Dobrý přehled o datech v tabulce dávají grafy. Zpravidla jeden sloupec tabulky obsahuje data pro vodorovnou osu a další sloupce pro svislou osu.

```
df.plot(x="Year")
```

```
<Axes: xlabel='Year'>
```



Někdy je vhodné ručně nastavit typ dat ve sloupcích. Například roky neni vhodné brát jako čísla, ale jako časový údaj. Popisky os jsou potom rozumnější.

```
df.dtypes
```

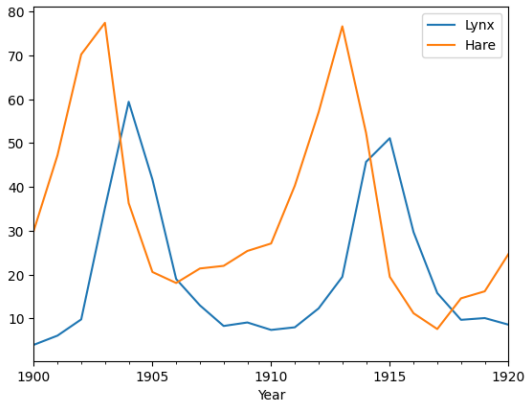
```
Year      int64
Lynx    float64
Hare    float64
dtype: object
```

```
# Převod čísel pro letopočty na datový␣
↪typ datetime.
df.Year = pd.to_datetime(df.Year,␣
↪format='%Y')
df.dtypes
```

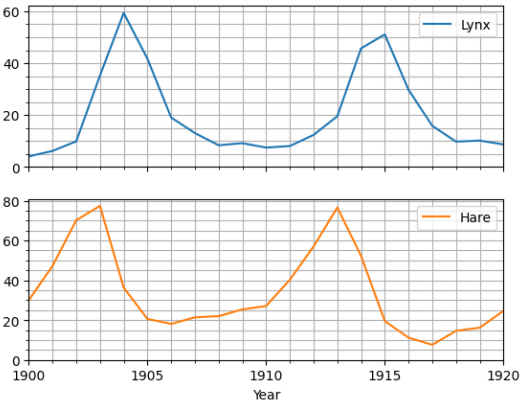
```
Year    datetime64[ns]
Lynx           float64
Hare           float64
dtype: object
```

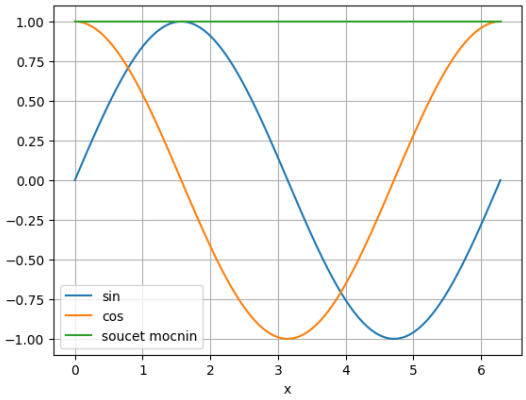
```
df.plot(x="Year")
```

```
<Axes: xlabel='Year'>
```




```
from matplotlib.ticker import␣
↪MultipleLocator

ax = df.plot(x="Year", subplots = True)
for a in ax:
    a.set(ylim=(0,None))
    a.grid(which='both')
    a.yaxis.set_minor_
↪locator(MultipleLocator(5))
```





31.3 Tvorba tabulek

Možností jak vytvořit tabulku je celá řada. Taždá možnost se hodí v jiné situaci. V zásadě můžeme buď přidávat sloupec po sloupci, nebo sestavit nějakou strukturu obsahující všechna data a z nich poté sestavit tabulku. Pokud uvedená struktura neumí pracovat se jmény sloupců. je nutné jména sloupců nastavit po vytvoření tabulky.

31.3.1 Možnost 1

Je možné nejprve vytvořit prázdnou tabulku a po sloupcích připojovat data. Sloupce musí být stejně dlouhé. Jméno sloupce definujeme při jeho vzniku.

```
x = np.linspace(0,2*np.pi,100)
df = pd.DataFrame()
df["x"] = x
df["sin"] = np.sin(x)
df["cos"] = np.cos(x)
df["soucet mocnin"] = ( df["sin"] )**2␣
↪+ ( df["cos"] )**2
df.plot(x="x", grid=True)
df.sample(5)
```

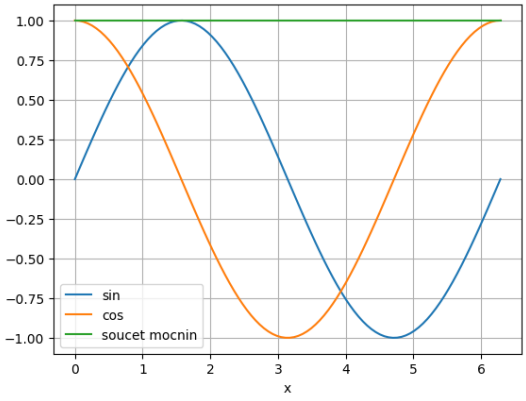
```
           x       sin       cos ␣
↪soucet mocnin
47  2.982926  0.158001 -0.987439    ␣
↪       1.0
26  1.650129  0.996855 -0.079250    ␣
↪       1.0
83  5.267721 -0.849725  0.527225    ␣
↪       1.0
68  4.315723 -0.922354 -0.386345    ␣
↪       1.0
96  6.092786 -0.189251  0.981929    ␣
↪       1.0
```

31.3.2 Možnost 2

Další možnost je načíst data ze seznamu. Pokud jsou v seznamu data po řádcích, je nutné je převést do sloupců transponováním. Poté nastavíme jména sloupců.

```
x = np.linspace(0,2*np.pi,100)
data = [
    x,
    np.sin(x),
    np.cos(x),
    np.sin(x)**2+np.cos(x)**2
    ]
df = pd.DataFrame(data).T
df.columns = ["x","sin","cos","soucet␣
↪mocnin"]
df.plot(x="x", grid=True)
df.sample(5)
```

```
           x       sin       cos ␣
↪soucet mocnin
98  6.219719 -0.063424  0.997987    ␣
↪       1.0
18  1.142397  0.909632  0.415415    ␣
↪       1.0
12  0.761598  0.690079  0.723734    ␣
↪       1.0
81  5.140788 -0.909632  0.415415    ␣
↪       1.0
97  6.156252 -0.126592  0.991955    ␣
↪       1.0
```

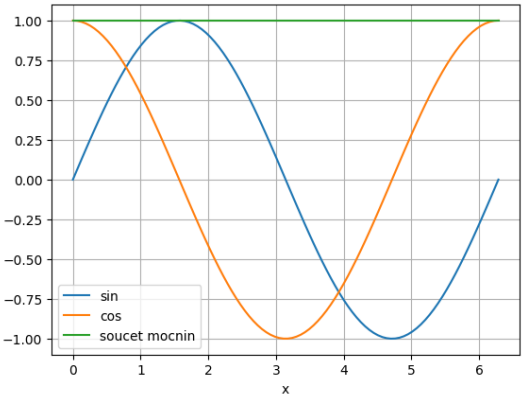


31.3.3 Možnost 3

Jiná možnost je použitím datového typu dictionary. Jména sloupců nastavujeme při tvorbě dictionary. Data do dictionary je možné nastavit najednou, nebo položku po položce.

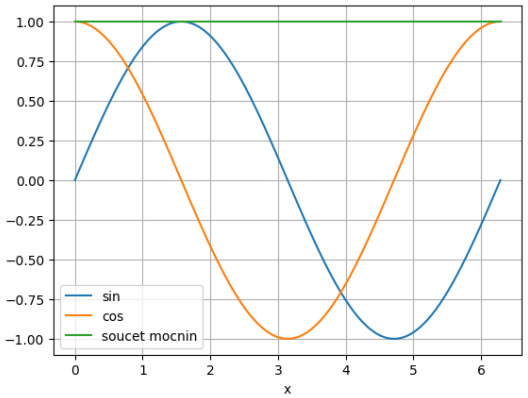
```
x = np.linspace(0,2*np.pi,100)
data = {
    "x":x,
    "sin":np.sin(x),
    "cos":np.cos(x),
    "soucet mocnin":np.sin(x)**2+np.
 ↪cos(x)**2
}
df = pd.DataFrame(data)
df.plot(x="x", grid=True)
df.sample(5)
```

```
           x       sin       cos ␣
 ↪soucet mocnin
7   0.444266  0.429795  0.902927    ␣
 ↪       1.0
82  5.204254 -0.881453  0.472271    ␣
 ↪       1.0
66  4.188790 -0.866025 -0.500000    ␣
 ↪       1.0
10  0.634665  0.592908  0.805270    ␣
 ↪       1.0
89  5.648520 -0.592908  0.805270    ␣
 ↪       1.0
```




```
x = np.linspace(0,2*np.pi,100)
data = {}
data["x"] = x
data["sin"] = np.sin(x)
data["cos"] = np.cos(x)
data["soucet mocnin"]=np.sin(x)**2+np.
 ↪cos(x)**2
df = pd.DataFrame(data)
df.plot(x="x", grid=True)

df.sample(5)
```

```
           x       sin       cos ␣
 ↪soucet mocnin
4   0.253866  0.251148  0.967949    ␣
 ↪       1.0
82  5.204254 -0.881453  0.472271    ␣
 ↪       1.0
80  5.077321 -0.934148  0.356886    ␣
 ↪       1.0
78  4.950388 -0.971812  0.235759    ␣
 ↪       1.0
74  4.696522 -0.999874 -0.015866    ␣
 ↪       1.0
```



KAPITOLA 32

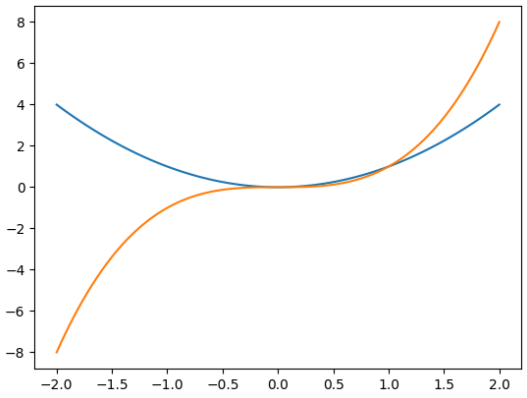
Obrázky, grafy

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
x = np.linspace(-2,2,100)
y = x**2
z = x**3
```

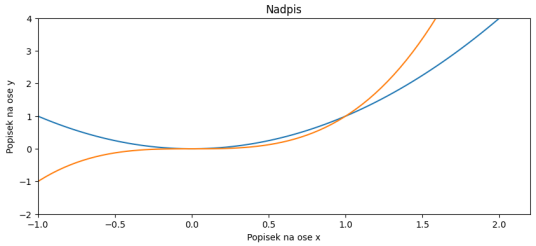
```
plt.plot(x,y)
plt.plot(x,z)
```

```
[<matplotlib.lines.Line2D at␣
 ↪0x7fcb8bca6e90>]
```

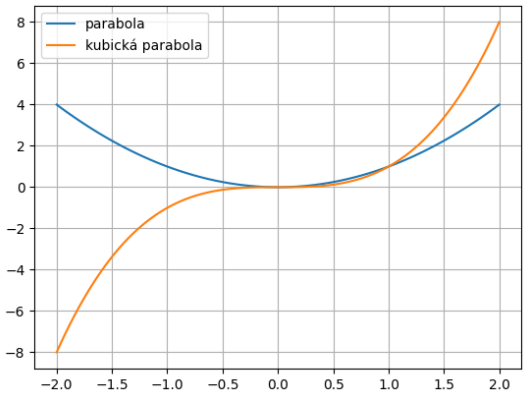


```
fig, ax = plt.subplots(1, figsize = (10,
 ↪4))
ax.plot(x,y)
ax.plot(x,z)
ax.set(
    title="Nadpis",
    xlabel="Popisek na ose x",
    ylabel="Popisek na ose y",
    xlim = (-1,None),
    ylim = (-2,4)
)
```

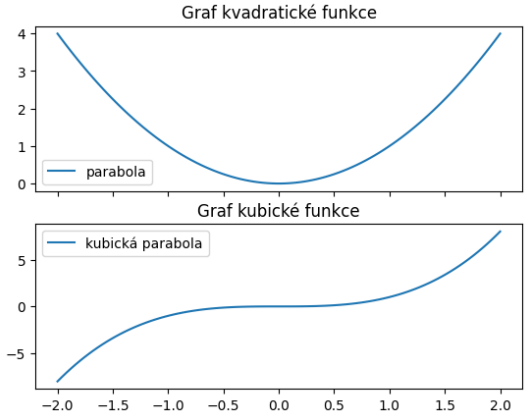
```
[Text(0.5, 1.0, 'Nadpis'),
 Text(0.5, 0, 'Popisek na ose x'),
 Text(0, 0.5, 'Popisek na ose y'),
 (-1.0, 2.2),
 (-2.0, 4.0)]
```

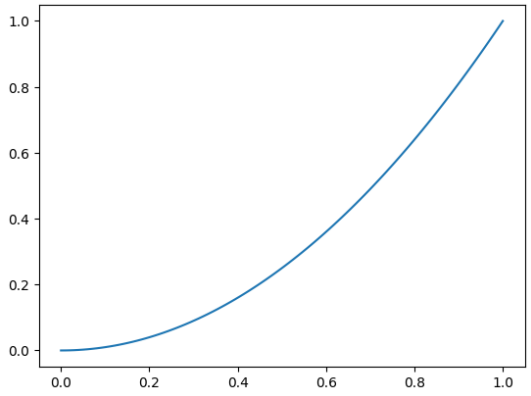



```
plt.plot(x,y,label="parabola")
plt.plot(x,z,label="kubická parabola")
plt.legend()
plt.grid()
```




```
fig,ax = plt.subplots(2,1,sharex=True)
ax[0].plot(x,y,label="parabola")
ax[1].plot(x,z,label="kubická parabola")
for i,j in zip(ax,["kvadratické",
 ↪"kubické"]):
    i.legend()
    i.set(title="Graf "+j+" funkce")
```



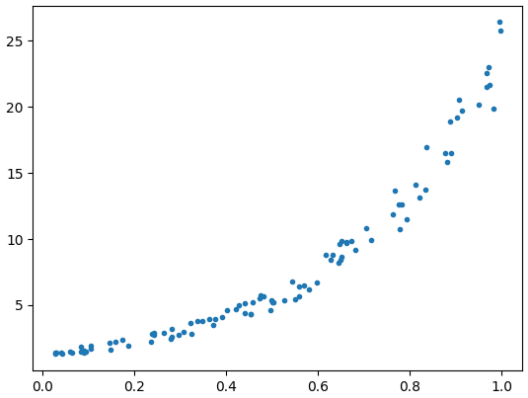


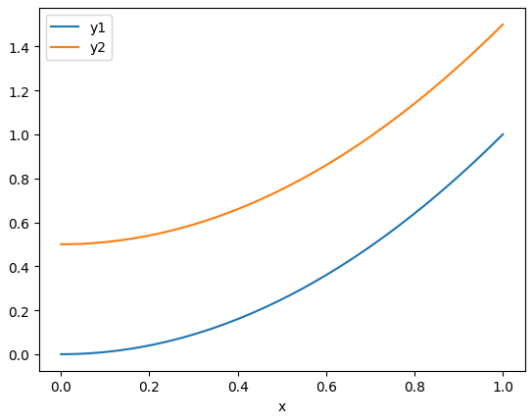
```
pocet = 100
x = np.random.random(pocet)
sum = np.random.random(pocet)
y = np.exp(3*x)*(1+0.4*sum)
fig, ax = plt.subplots(1)
plt.plot(x,y,".")
#ax.set(yscale="log")
```

```
x = np.linspace(0,1,100)
df = pd.DataFrame()
df["x"] = x
df["y1"] = x**2
df["y2"] = x**2+.5
fig,ax = plt.subplots(1)
df.plot(x="x", ax=ax)
# ax.set(
#     xlim=(None,None),
#     ylim=(-1,None),
#     title="Můj graf",
#     xlabel="osa x",
#     ylabel="osa y",
# )
# plt.grid()
```

```
[<matplotlib.lines.Line2D at␣
↪0x7fcb89916ed0>]
```

```
<Axes: xlabel='x'>
```






```
x = np.linspace(0,1,100)
y = x**2
fig,ax = plt.subplots(1)
ax.plot(x,y)
# ax.set(
#     xlim=(None,None),
#     ylim=(-1,None),
#     title="Můj graf",
#     xlabel="osa x",
#     ylabel="osa y",
# )
# plt.grid()
```

```
[<matplotlib.lines.Line2D at␣
↪0x7fcb8994af10>]
```

KAPITOLA 33

Řešení diferenciální rovnice

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.integrate import solve_ivp
```

Pravou stranu rovnice je výhodné mít definovánu jako samostatnou funkci. Má první argument čas, druhý argument derivovanou veličinu, může mít parametry a tyto parametry mohou mít implicitní hodnoty. (Implicitní hodnoty se použijí, pokud není zadána jiná hodnota parametru.)

Následující funkce je pravou stranou logistické rovnice

$$\frac{\mathrm{d}y}{\mathrm{d}t} = ry\left(1 - \frac{y}{K}\right).$$

```
def rovnice(t,y,r=1,K=1):
    return r*y*(1-y/K)
```

Nastavení počátečních podmínek je vhodné udělat mimo příkaz řešící rovnici, aby se případné modifikace dělaly vždy na jednom místě. Je vhodné sadu příkazů rozdělit do více buněk, aby se nepočítalo to, co není nutné počítat znovu.

```
pocatecni_podminka = [0.1]
meze = [0,10]
t = np.linspace(0,10,100)
```

33.1 Varianta 1: Chceme funkční předpis

Při volání funkce `solve_ivp` rovnou použijeme metodu `sol`. Výsledkem je funkce, do keré můžeme dosazovat libovolný čas mezi dolní a horní mezí a získáme hodnotu řešení v daném čase. Můžeme také dosazovat několik hodnot času současně a získáme funkční hodnotu pro každý časový okamžik.

```
reseni_spojite = solve_ivp(
                    rovnice,
                    meze,
                    pocatecni_podminka,
                    dense_output=True,
                    ).sol
```

```
reseni_spojite(1)
```

```
array([0.23199499])
```

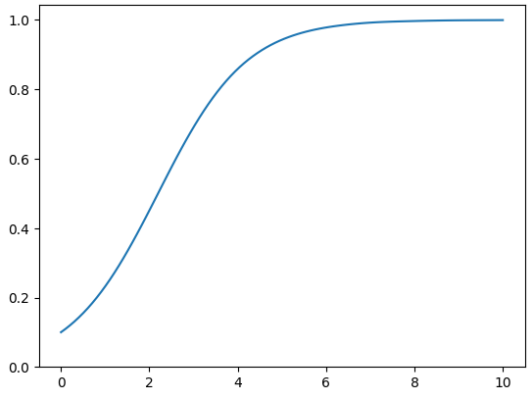
```
reseni_spojite(t).shape
```

```
(1, 100)
```

```
reseni_spojite(t)[0].shape
```

```
(100,)
```

```
plt.plot(t,reseni_spojite(t)[0])
ax = plt.gca()
ax.set(ylim=(0,None));
```



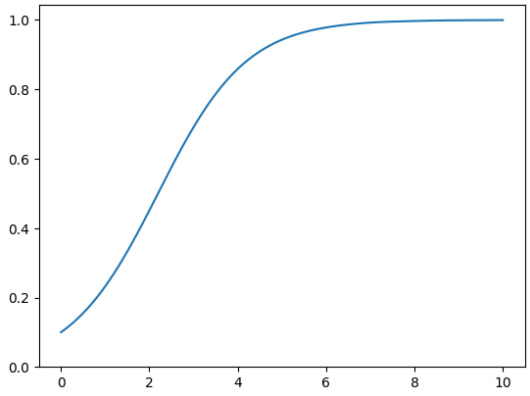
33.2 Varianta 2: Chceme data pro předem známé časy

Hodnoty času, pro které se má určit funkční hodnota řešení, se zadají při volání příkazu `solve_ivp` pomocí parametru `t_eval`.

```
reseni_data = solve_ivp(
                    rovnice,
                    meze,
                    pocatecni_podminka,
                    t_eval=t,
                    ).y
reseni_data.shape
```

```
(1, 100)
```

```
plt.plot(t,reseni_data[0])
ax = plt.gca()
ax.set(ylim=(0,None));
```



33.3 Různé hodnoty parametrů

Pokud chceme určit chování řešení pro různé hodnoty parametrů, je nejvýhodnější řešit rovnici v každém případě pro stejné hodnoty času a výsledek uspořádat jako sloupce tabulky. Vytvoříme tedy seznam obsahující jednotlivé funkční hodnoty, sestavíme z nich tabulku a přidáme sloupec s časem. Poté můžeme dělat cokoliv, například řešení vykreslit.

```
parametry_r = [1,0.5,0.25]
meze_2 = [0,20]
t_2 = np.linspace(0,20,100)
reseni_data = [
                  solve_ivp(
                     lambda t,x:␣
 ↪rovnice(t,x,r=r),
                     meze_2,
                     pocatecni_podminka,
                     t_eval=t_2,
                     ).y[0]
                  for r in parametry_r
              ]
reseni_data = pd.DataFrame(np.
 ↪array(reseni_data).T,
 ↪columns=parametry_r)
reseni_data["t"] = t_2
reseni_data
```

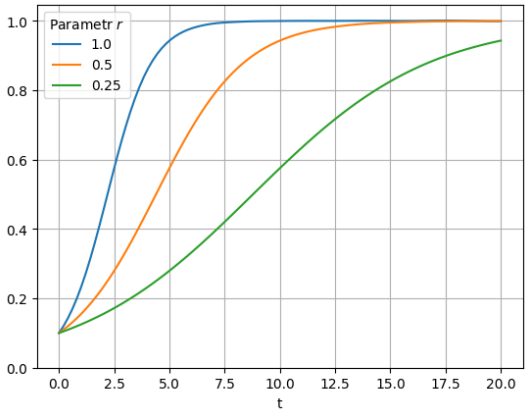
```
          1.0       0.5       0.25     ␣
 ↪     t
0    0.100000  0.100000  0.100000   0.
 ↪000000
1    0.119702  0.109465  0.104638   0.
 ↪202020
2    0.142664  0.119707  0.109465   0.
 ↪404040
3    0.169211  0.130767  0.114487   0.
 ↪606061
4    0.199558  0.142683  0.119707   0.
 ↪808081
..        ...       ...       ...     ␣
 ↪    ...
95   0.999366  0.999297  0.931092  19.
 ↪191919
96   0.999170  0.999342  0.934262  19.
 ↪393939
97   0.999041  0.999394  0.937297  19.
 ↪595960
98   0.999009  0.999451  0.940201  19.
 ↪797980
99   0.999110  0.999504  0.942978  20.
 ↪000000

[100 rows x 4 columns]
```

```
ax = reseni_data.plot(x="t")
plt.legend(title=r"Parametr $r$")
ax.set(ylim=(0,None))
```

(continues on next page)

(pokračujte na předchozí stránce)

```
ax.grid()
```



KAPITOLA 34

Tabulky s None

S tabulkami se pracuje pomocí knihovny Pandas. Nejsnazší je tvořit tabulky z matice nebo po sloupcích. Přitom musí mít každý sloupec stejný počet dat. Potom jsou všechny sloupce stejně dlouhé. Toto se však nedá zařídit vždy, někdy je nutno do tabulky přidat sloupec, který je kratší. Řešením je přidat do tabulky sloupec s libovolnými hodnotami a poté nahradit ta data, která máme k dispozici. Vhodnou výplní nedefinovaných dat jsou nuly nebo hodnoty `None` nebo `np.nan`.

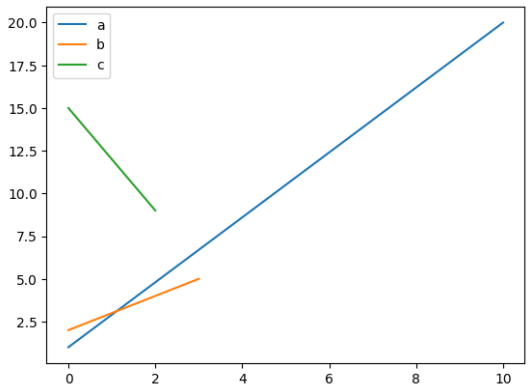
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

N = 11
df = pd.DataFrame()
df["a"] = np.linspace(1,20,N)

# založíme prázdný sloupec a potom jeho␣
 ↪začátek přepíšeme daty
df["b"] = np.nan
data = np.array([2,3,4,5])
df.loc[:len(data)-1,"b"] = data

# jiná varianta je nejprve doplnit data␣
 ↪a potom zakládat sloupec
data = np.array([15,12,9])
doplnena_data = np.append(data,np.
 ↪full(N-len(data),np.nan))
df["c"] = doplnena_data

df.plot()
df
```

```
      a    b     c
0   1.0  2.0  15.0
1   2.9  3.0  12.0
2   4.8  4.0   9.0
3   6.7  5.0   NaN
4   8.6  NaN   NaN
5  10.5  NaN   NaN
```

(continues on next page)

(pokračujte na předchozí stránce)

```
6   12.4  NaN   NaN
7   14.3  NaN   NaN
8   16.2  NaN   NaN
9   18.1  NaN   NaN
10  20.0  NaN   NaN
```



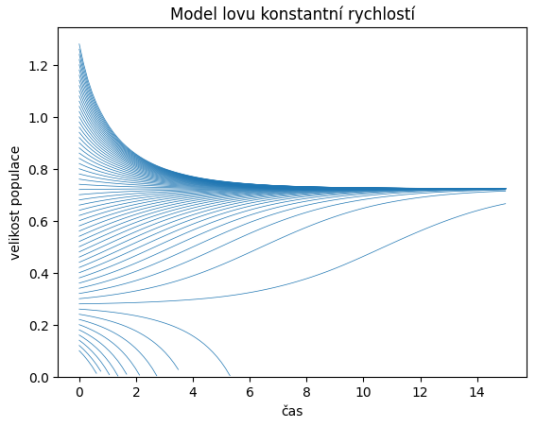
Ukázkou je rovnice konstantního lovu v logistické rovnici, kdy pro některé počáteční podmínky je populace přelovená a dochází k její destrukci. Řešení se nemusí zastavit dosažením cílového času, ale při poklesu velikosti populace na nulu. Pokud se snažíme zapsat dat do jedné tabulky, musíme neobsazená místa zaplnit nedefinovanými hodnotami.

```
### Příprava funkcí a parametrů
pocatecni_podminky = np.round(np.
 ↪arange(0.1,1.3,0.02),2)  # počáteční␣
 ↪podmínka nebo podmínky
meze = [0,15]  # interval, na kterém␣
 ↪hledáme řešení
N = 100 # počet dělících bodů

def model(t, N, r=1,K=1,h=0.2):
    return  r*N*(1-N/K) - h

def destrukce_populace(t,x,r=1,K=1,h=0.
 ↪15):  # Pokud x klesne na nulu,␣
```

(continues on next page)

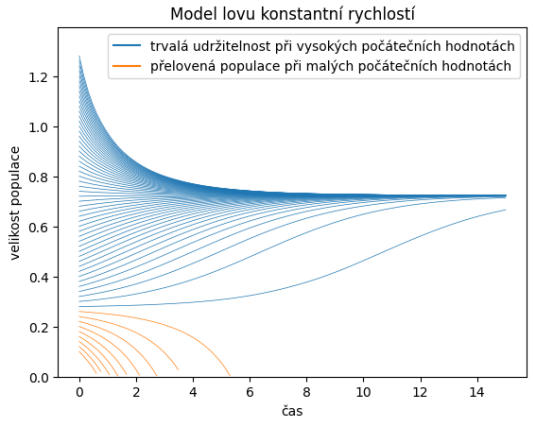
(pokračujte na předchozí stránce)

```
 ↪zastavíme výpočet
    return x
destrukce_populace.terminal = True

### Řešení modelu
t=np.linspace(*meze, N)  # definiční␣
 ↪obor, v~těchto bodech budeme hledat␣
 ↪řešení
df = pd.DataFrame()
df["t"] = t

for pocatecni_podminka in pocatecni_
 ↪podminky:
    reseni = solve_ivp(
                        model,
                        meze,
                        [pocatecni_
 ↪podminka],
                        t_eval=t,
                        events=destrukce_
 ↪populace,
                        )
    data = reseni.y[0]
    if len(data)==N:
        df[pocatecni_podminka] = data
    else:
        df[pocatecni_podminka] = None
        df.loc[:len(data)-1,pocatecni_
 ↪podminka] = data
        # jina varianta je nejprve␣
 ↪doplnit data a potom zakladat sloupec
        # doplnena_data = np.
 ↪append(data,np.full(N-len(data),np.
 ↪nan))
        # df[pocatecni_podminka] =␣
 ↪doplnena_data

### Vizualizace řešení

ax = df.plot(x="t",lw=0.5,legend=False,
 ↪color="C0")
ax.set(
    ylim = (0,None),
    title = "Model lovu konstantní␣
 ↪rychlostí",
    xlabel="čas",
    ylabel="velikost populace",
);
```



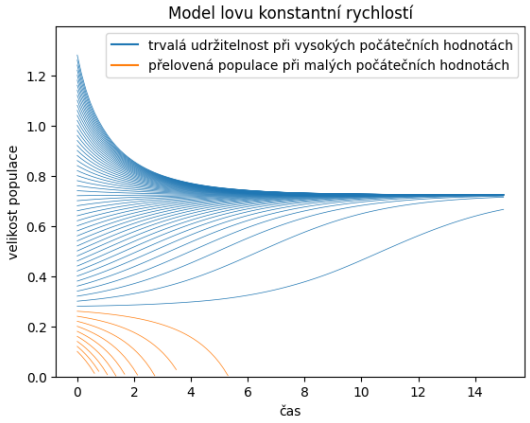
Můžeme i barevně rozlišit, které křivky vedou k vymření přelovené populace.

```
fig,ax = plt.subplots()
plt.plot([0,0],[-1,-1])
plt.plot([0,0],[-1,-1])
for pocatecni_podminka in pocatecni_
 ↪podminky:
    data = np.array(df[pocatecni_
 ↪podminka])
    if pd.isna(data).any():
        barva = "C1"
    else:
        barva = "C0"
    ax.plot(t,data, color=barva, lw=0.5)

ax.set(
    ylim = (0,None),
    title = "Model lovu konstantní␣
 ↪rychlostí",
    xlabel="čas",
    ylabel="velikost populace",
)
plt.legend(["trvalá udržitelnost při␣
 ↪vysokých počátečních hodnotách",
 ↪"přelovená populace při malých␣
 ↪počátečních hodnotách"]);
```



Místo cyklu přes všechna řešení můžeme vykreslit jednou barvou sloupce s nedefinovanými hodnotami a druhou barvou sloupce bez nich. Toto bude efektivní zejména pro rozsáhlé soubory dat.

```
fig,ax = plt.subplots()
plt.plot([0,0],[-1,-1])
plt.plot([0,0],[-1,-1])

# viz https://stackoverflow.com/
↪questions/47414848/pandas-select-all-
↪columns-without-nan
df[df.columns[~df.isnull().any()]].
↪plot(color="C0", ax=ax, x="t", lw=0.5)
df[list(df.columns[df.isnull().any()])+[
↪"t"]].plot(color="C1", ax=ax, x="t",␣
↪lw=0.5)

ax.set(
    ylim = (0,None),
    title = "Model lovu konstantní␣
↪rychlostí",
    xlabel="čas",
    ylabel="velikost populace",
)
plt.legend(["trvalá udržitelnost při␣
↪vysokých počátečních hodnotách",
↪"přelovená populace při malých␣
↪počátečních hodnotách"]);
```




```
df
```

```
             t       0.1       0.12    ␣
↪  0.14      0.16      0.18       0.
↪2   \
0    0.000000       0.1       0.12    ␣
↪  0.14      0.16      0.18       0.
↪2
1    0.151515  0.082256  0.104823  0.
↪127245  0.149523  0.171659  0.
↪193653
2    0.303030  0.062067   0.08767  0.
↪112927  0.137843  0.162424  0.
↪186676
3    0.454545  0.039067   0.06827  0.
↪096848  0.124819  0.152198  0.
↪179003
4    0.606061  0.012689  0.046222  0.
↪078731  0.110263  0.140859  0.
```

(continues on next page)

(pokračujte na předchozí stránce)

```
↪170559
..        ...       ...       ...    ␣
↪   ...       ...       ...       ..
↪.
95  14.393939       None       None    ␣
↪  None       None       None      ␣
↪None
96  14.545455       None       None    ␣
↪  None       None       None      ␣
↪None
97  14.696970       None       None    ␣
↪  None       None       None      ␣
↪None
98  14.848485       None       None    ␣
↪  None       None       None      ␣
↪None
99  15.000000       None       None    ␣
↪  None       None       None      ␣
↪None

        0.22      0.24      0.26  ...
↪       1.1      1.12      1.14    ␣
↪ 1.16  \
0       0.22      0.24      0.26  ...
↪  1.100000  1.120000  1.140000  1.
↪160000
1   0.215508  0.237225  0.258806  ...
↪  1.056946  1.073683  1.090328  1.
↪106880
2   0.210605  0.234217   0.25752  ...
↪  1.020253  1.034409  1.048422  1.
↪062293
3    0.20525  0.230955  0.256135  ...
↪  0.988909  1.001039  1.013003  1.
↪024804
4   0.199401  0.227417  0.254644  ...
↪  0.962271  0.972819  0.983196  0.
↪993406
..       ...       ...       ...  ...
↪       ...       ...       ...    ␣
↪  ...
95      None      None      None  ...
↪  0.724021  0.724059  0.724068  0.
↪724063
96      None      None      None  ...
↪  0.724007  0.724039  0.724038  0.
↪724033
97      None      None      None  ...
↪  0.723991  0.724016  0.724009  0.
↪724005
98      None      None      None  ...
↪  0.723972  0.723990  0.723983  0.
↪723979
99      None      None      None  ...
↪  0.723951  0.723965  0.723958  0.
↪723954

        1.18       1.2      1.22    ␣
↪ 1.24      1.26      1.28
0   1.180000  1.200000  1.220000  1.
↪240000  1.260000  1.280000
```

(continues on next page)

(pokračujte na předchozí stránce)

```
1   1.123341  1.139711  1.155991  1.
 ↪172181  1.188284  1.204298
2   1.076024  1.089618  1.103077  1.
 ↪116403  1.129598  1.142664
3   1.036447  1.047935  1.059272  1.
 ↪070461  1.081506  1.092409
4   1.003455  1.013345  1.023082  1.
 ↪032668  1.042108  1.051405
..       ...       ...       ...    ␣
 ↪  ...       ...       ...
95  0.724059  0.724058  0.724057  0.
 ↪724058  0.724059  0.724061
96  0.724030  0.724028  0.724028  0.
 ↪724028  0.724030  0.724032
97  0.724002  0.724001  0.724000  0.
 ↪724001  0.724002  0.724005
98  0.723976  0.723975  0.723974  0.
 ↪723975  0.723977  0.723979
99  0.723952  0.723951  0.723950  0.
 ↪723951  0.723952  0.723955

[100 rows x 61 columns]
```

Začátek a konec tabulky s jedním předčasně končícím řešením.

```
df.loc[:10,[0.24,0.5,0.8]]
```

```
        0.24       0.5       0.8
0       0.24  0.500000  0.800000
1   0.237225  0.507573  0.794205
2   0.234217  0.515129  0.788898
3   0.230955  0.522650  0.784023
4   0.227417  0.530120  0.779548
5    0.22358  0.537521  0.775442
6   0.219411  0.544839  0.771675
7   0.214874  0.552058  0.768219
8   0.209926  0.559164  0.765046
9   0.204521  0.566144  0.762127
10  0.198605  0.572986  0.759438
```

```
df.loc[N-10:,[0.24,0.5,0.8]]
```

```
    0.24       0.5       0.8
90  None  0.722593  0.723826
91  None  0.722658  0.723811
92  None  0.722719  0.723798
93  None  0.722775  0.723785
94  None  0.722828  0.723774
95  None  0.722876  0.723763
96  None  0.722922  0.723753
97  None  0.722965  0.723743
98  None  0.723006  0.723734
99  None  0.723045  0.723726
```

KAPITOLA 35

Tabulky s víceúrovňovými nadpisy sloupců

Rovnice

$$\frac{\mathrm{d}n}{\mathrm{d}t} = rn\left(1 - \frac{n}{K}\right) - h$$

popisuje vývoj populace o velikosti n v prostředí s nosnou kapacitou K. Budeme pro různou intenzitu lovu vždy sledovat řešení pro několik počátečních podmínek. Všechno zaznamenáme do tabulky a data z tabulky poté vykreslíme. Sloupec bude identifikován pomocí trojice hodnot, udávajících že jde o velikosti populací (a ne čas), dále intenzitu lovu a počáteční podmínku.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.integrate import solve_ivp
```

```
pocatecni_podminka = [0.4]
meze = [0,10]

def rovnice(t, x, r=1, K=1, h=0):
    return r*x*(1-x/K) - h

def destrukce_populace(t,x,r=1,K=1,
 ↪h=0):  # Pokud x klesne na nulu,␣
 ↪zastavíme výpočet
    return x
destrukce_populace.terminal = True

t=np.linspace(*meze,101)
seznam_h = [0,0.1,0.15,0.2,0.25,0.28]
seznam_pp = np.round(np.arange(0.1,1.2,
 ↪0.05),2)

### Definice tabulky s~víceúrovňovými␣
 ↪nadpisy sloupců, MultiIndex
### https://pandas.pydata.org/docs/user_
 ↪guide/advanced.html
iterables = [["data"],seznam_h,seznam_
 ↪pp]
my_index = pd.MultiIndex.from_
 ↪product(iterables, names=['typ','lov',
```

(continues on next page)

(pokračujte na předchozí stránce)

```
 ↪ 'poč.podm.'])
df = pd.DataFrame(columns=my_index,␣
 ↪dtype='float64')

df["čas"] = t
df
```

```
typ        data                                ␣
 ↪                              ...            ␣
 ↪     \
lov         0.0                                ␣
 ↪                              ... 0.28
poč.podm.   0.1 0.15 0.2 0.25 0.3 0.
 ↪35 0.4 0.45 0.5 0.55   ... 0.75 0.8␣
 ↪0.85
0           NaN  NaN NaN  NaN NaN ␣
 ↪NaN NaN  NaN NaN  NaN  ...  NaN␣
 ↪NaN  NaN
1           NaN  NaN NaN  NaN NaN ␣
 ↪NaN NaN  NaN NaN  NaN  ...  NaN␣
 ↪NaN  NaN
2           NaN  NaN NaN  NaN NaN ␣
 ↪NaN NaN  NaN NaN  NaN  ...  NaN␣
 ↪NaN  NaN
3           NaN  NaN NaN  NaN NaN ␣
 ↪NaN NaN  NaN NaN  NaN  ...  NaN␣
 ↪NaN  NaN
4           NaN  NaN NaN  NaN NaN ␣
 ↪NaN NaN  NaN NaN  NaN  ...  NaN␣
 ↪NaN  NaN
..          ...  ...  ..  ...  ..  ...
 ↪  ..  ...  ..  ...  ...  ...  ..  .
 ↪..
96          NaN  NaN NaN  NaN NaN ␣
 ↪NaN NaN  NaN NaN  NaN  ...  NaN␣
 ↪NaN  NaN
97          NaN  NaN NaN  NaN NaN ␣
 ↪NaN NaN  NaN NaN  NaN  ...  NaN␣
 ↪NaN  NaN
98          NaN  NaN NaN  NaN NaN ␣
 ↪NaN NaN  NaN NaN  NaN  ...  NaN␣
```

(continues on next page)

(pokračujte na předchozí stránce)

```
 ↪NaN  NaN
99          NaN  NaN NaN  NaN NaN ␣
 ↪NaN NaN  NaN NaN  NaN  ...  NaN␣
 ↪NaN  NaN
100         NaN  NaN NaN  NaN NaN ␣
 ↪NaN NaN  NaN NaN  NaN  ...  NaN␣
 ↪NaN  NaN

typ                                ␣
 ↪  čas
lov
poč.podm. 0.9 0.95 1.0 1.05 1.1 1.15
0          NaN  NaN NaN  NaN NaN  NaN␣
 ↪  0.0
1          NaN  NaN NaN  NaN NaN  NaN␣
 ↪  0.1
2          NaN  NaN NaN  NaN NaN  NaN␣
 ↪  0.2
3          NaN  NaN NaN  NaN NaN  NaN␣
 ↪  0.3
4          NaN  NaN NaN  NaN NaN  NaN␣
 ↪  0.4
..          ..  ...  ..  ...  ..  ...␣
 ↪  ...
96         NaN  NaN NaN  NaN NaN  NaN␣
 ↪  9.6
97         NaN  NaN NaN  NaN NaN  NaN␣
 ↪  9.7
98         NaN  NaN NaN  NaN NaN  NaN␣
 ↪  9.8
99         NaN  NaN NaN  NaN NaN  NaN␣
 ↪  9.9
100        NaN  NaN NaN  NaN NaN  NaN␣
 ↪ 10.0

[101 rows x 133 columns]
```

Dále ve dvojitém cyklu pro každou počáteční podmínku a úroveň lovu najdeme řešení. Hledání řešení ukončíme při dosažení konce časového intervalu, nebo při dosažení nulové hodnoty pro velikost populace. Proto jsme tabulku založili rovnou se sloupci, které mají nedefinované hodnoty `np.nan`. Nyní vyměníme tolik položek ze začátku sloupce, kolik máme k dispozici vypočtených hodnot.

Tabulku vytiskneme, pro lepší přehlednost naležato.

```
for h in seznam_h:
    for pp in seznam_pp:
        reseni = solve_ivp(
                    lambda t,x:rovnice(t,
 ↪x,h=h),
                    meze,
                    [pp],
                    t_eval=t,
                    events=destrukce_
 ↪populace,
                    )
        a = np.array(reseni.y.T)
        df.loc[:len(a)-1,('data',h,pp)]␣
```

(continues on next page)

(pokračujte na předchozí stránce)

```
 ↪= a  # poté hodnoty zaměníme za␣
 ↪vypočtené od začátku až po konec␣
 ↪řešení
df.T   # vytiskneme tabulku naležato  ␣
 ↪
```

```
                        0         1   ␣
 ↪     2         3         4        ␣
 ↪5    \
typ  lov  poč.podm.
data 0.0  0.1         0.10  0.109367 ␣
 ↪0.119490  0.130409  0.142170  0.
 ↪154811
          0.15        0.15  0.163201 ␣
 ↪0.177317  0.192371  0.208383  0.
 ↪225362
          0.2         0.20  0.216481 ␣
 ↪0.233921  0.252314  0.271641  0.
 ↪291872
          0.25        0.25  0.269214 ␣
 ↪0.289339  0.310332  0.332133  0.
 ↪354672
          0.3         0.30  0.321410 ␣
 ↪0.343605  0.366510  0.390029  0.
 ↪414061
...                     ...       ... ␣
 ↪     ...       ...       ...      ␣
 ↪...
     0.28 1.0         1.00  0.973331 ␣
 ↪0.949011  0.926629  0.906047  0.
 ↪887138
          1.05        1.05  1.018480 ␣
 ↪0.989968  0.963915  0.940152  0.
 ↪918513
          1.1         1.10  1.063204 ␣
 ↪1.030200  1.000301  0.973271  0.
 ↪948868
          1.15        1.15  1.107508 ␣
 ↪1.069714  1.035770  1.005349  0.
 ↪978114
čas                   0.00  0.100000 ␣
 ↪0.200000  0.300000  0.400000  0.
 ↪500000


                           6        ␣
 ↪7         8         9    ...      ␣
 ↪91   \
typ  lov  poč.podm.                 ␣
 ↪                         ...
data 0.0  0.1         0.168360  0.
 ↪182841  0.198269  0.214654  ...  0.
 ↪998996
          0.15        0.243304  0.
 ↪262194  0.282004  0.302695  ...  0.
 ↪999423
          0.2         0.312962  0.
 ↪334858  0.357490  0.380778  ...  0.
 ↪999461
          0.25        0.377873  0.
 ↪401645  0.425892  0.450506  ...  0.
```

(continues on next page)

(pokračujte na předchozí stránce)

```
 ↪999576
          0.3        0.438501  0.
 ↪463241  0.488165  0.513157  ...  0.
 ↪999715
...                          ...       .
 ↪..       ...       ...  ...       .
 ↪..
     0.28 1.0        0.869770  0.
 ↪853805  0.839105  0.825525  ...  0.
 ↪441710
          1.05       0.898820  0.
 ↪880888  0.864520  0.849509  ...  0.
 ↪446593
          1.1        0.926839  0.
 ↪906917  0.888824  0.872266  ...  0.
 ↪450776
          1.15       0.953714  0.
 ↪931781  0.911931  0.893763  ...  0.
 ↪455037
čas                  0.600000  0.
 ↪700000  0.800000  0.900000  ...  9.
 ↪100000

                             92       ␣
 ↪93        94        95        96   ␣
 ↪\
typ  lov  poč.podm.
data 0.0  0.1        0.999076  0.
 ↪999143  0.999201  0.999252  0.
 ↪999299
          0.15       0.999458  0.
 ↪999483  0.999501  0.999515  0.
 ↪999528
          0.2        0.999512  0.
 ↪999559  0.999601  0.999639  0.
 ↪999673
          0.25       0.999593  0.
 ↪999616  0.999648  0.999681  0.
 ↪999712
          0.3        0.999717  0.
 ↪999718  0.999720  0.999724  0.
 ↪999733
...                          ...       .
 ↪..       ...       ...       ...
     0.28 1.0        0.438351  0.
 ↪434949  0.431502  0.428006  0.
 ↪424460
          1.05       0.443288  0.
 ↪439944  0.436559  0.433129  0.
 ↪429653
          1.1        0.447519  0.
 ↪444226  0.440895  0.437522  0.
 ↪434106
          1.15       0.451865  0.
 ↪448657  0.445411  0.442123  0.
 ↪438792
čas                  9.200000  9.
 ↪300000  9.400000  9.500000  9.
 ↪600000

                             97       ␣
```

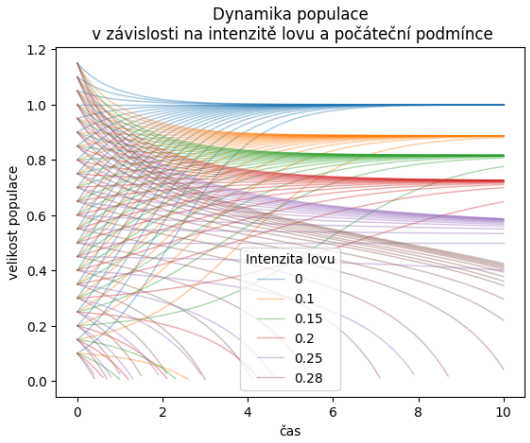
(continues on next page)

(pokračujte na předchozí stránce)

```
 ↪98        99         100
typ  lov  poč.podm.
data 0.0  0.1        0.999347  0.
 ↪999399  0.999455  0.999507
          0.15       0.999543  0.
 ↪999564  0.999595  0.999633
          0.2        0.999704  0.
 ↪999732  0.999757  0.999780
          0.25       0.999739  0.
 ↪999764  0.999786  0.999807
          0.3        0.999751  0.
 ↪999774  0.999796  0.999815
...                          ...       .
 ↪..       ...       ...
     0.28 1.0        0.420859  0.
 ↪417201  0.413482  0.409700
          1.05       0.426128  0.
 ↪422551  0.418919  0.415230
          1.1        0.430645  0.
 ↪427135  0.423575  0.419962
          1.15       0.435414  0.
 ↪431987  0.428507  0.424973
čas                  9.700000  9.
 ↪800000  9.900000  10.000000

[133 rows x 101 columns]
```

Na závěr tabulku vykreslíme. Například všechny křivky odpovídající stejné intenzitě lovu vykreslíme stejnou barvou.

```
for i,h in enumerate(seznam_h):
    plt.plot(df["čas"],df["data"][h],
 ↪color=f"C{i}",label=h,alpha=0.4,lw=1)

ax = plt.gca()
ax.set(
    title = "Dynamika populace\n␣
 ↪v~závislosti na intenzitě lovu a␣
 ↪počáteční podmínce",
    xlabel="čas",
    ylabel="velikost populace",
)
# Návod jak seskupit položky legendy je␣
 ↪na https://stackoverflow.com/
 ↪questions/26337493/pyplot-combine-
 ↪multiple-line-labels-in-legend
handles, labels = ax.get_legend_handles_
 ↪labels()
labels, ids = np.unique(labels, return_
 ↪index=True)
handles = [handles[i] for i in ids]
ax.legend(handles, labels, title=
 ↪"Intenzita lovu");
```



Na závěr ukázka, jak snadno z tabulky vybereme všechna řešení s počáteční podmínkou 0.2. Jde vidět, že tři hodnoty lovu vedou k destrukci populace v konečném čase.

```
df.loc[:,("data",slice(None),0.2)]
```

```
typ            data
lov             0.0       0.1       0.
↪15       0.2       0.25      0.28
poč.podm.       0.2       0.2       ␣
↪0.2       0.2       0.2       0.2
0          0.200000  0.200000  0.
↪200000  0.200000  0.200000  0.
↪200000
1          0.216481  0.206182  0.
↪201031  0.195877  0.190722  0.
↪187628
2          0.233921  0.212738  0.
↪202125  0.191493  0.180838  0.
↪174433
3          0.252314  0.219677  0.
↪203285  0.186822  0.170284  0.
↪160327
4          0.271641  0.227018  0.
↪204517  0.181849  0.159020  0.
↪145247
..              ...       ...       .
↪..       ...       ...       ...
96         0.999673  0.883717  0.
↪765607       NaN       NaN       ␣
↪NaN
97         0.999704  0.883966  0.
↪768472       NaN       NaN       ␣
↪NaN
98         0.999732  0.884200  0.
↪771186       NaN       NaN       ␣
↪NaN
99         0.999757  0.884423  0.
↪773758       NaN       NaN       ␣
↪NaN
100        0.999780  0.884635  0.
↪776194       NaN       NaN       ␣
↪NaN

[101 rows x 6 columns]
```

Část IV

Literatura a rejstřík

KAPITOLA 36

Literatura

KAPITOLA 37

Rejstřík

Literatura

[1] R.M.N. Alexander. *Optima for Animals*. Optima for Animals. Princeton University Press, 1996. ISBN 9780691027999.

[2] U. Alon. *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Chapman & Hall/CRC Computational Biology Series. CRC Press LLC, 2019. ISBN 9781439837177.

[3] M. Begon, C.R. Townsend, and J.L. Harper. *Ekologie: jedinci, populace a společenstva*. Vydavatelství Univerzity Palackého, 1997. ISBN 9788070676950.

[4] Hal Caswell. *Matrix population models*. Sinauer, 2. ed edition, 2001. ISBN 0878930965.

[5] Z. Došlá and P. Liška. *Matematika pro nematematické obory s aplikacemi v přírodních a technických vědách*. Grada, 2014. ISBN 9788024753225.

[6] D. Dykyjová. *Metody studia ekosystémů*. Academia, 1989.

[7] R. S. Etienne. A scrutiny of the levins metapopulation model. *Comments on Theoretical Biology*, 7:257–281, 2002. URL: https://core.ac.uk/download/pdf/29299949.pdf.

[8] A.C. Fowler, A.C. Fowler, D.G. Crighton, and M.J. Ablowitz. *Mathematical Models in the Applied Sciences*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 1997. ISBN 9780521467032.

[9] J. Kalas and Z. Pospíšil. *Spojité modely v biologii*. Masarykova univerzita, 2001.

[10] J. Kalas and M. Ráb. *Obyčejné diferenciální rovnice*. Masarykova univerzita, Brno, vyd. 3. edition, 2012. ISBN 9788021058156.

[11] P. Kovář. *Ekosystémová a krajinná ekologie*. Karolinum, 2014. ISBN 9788024627885. URL: https://www.natur.cuni.cz/biologie/botanika/studium/vybrane-predmety/prirucky-pavla-kovare.

[12] M. Monagan. Using Leslie matrices as the application of eigenvalues and eigenvectors in a first course in linear algebra. In Jürgen Gerhard and Ilias Kotsireas, editors, *Maple in Mathematics Education and Research*, 279–291. Cham, 2020. Springer International Publishing. URL: https://link.springer.com/chapter/10.1007/978-3-030-41258-6_21.

[13] J. D. Murray. *Mathematical Biology*. Volume 19 of Biomathematics. Springer, 1989. URL: https://link.springer.com/book/10.1007/978-3-662-08539-4.

[14] J. D. Murray. *Mathematical Biology I. An Introduction*. Volume 17 of Interdisciplinary Applied Mathematics. Springer, New York, 3 edition, 2002. URL: https://link.springer.com/book/10.1007/b98868, doi:10.1007/b98868.

[15] E. Tkadlec. *Populační ekologie: Struktura a růst populací*. Univerzita Palckého v Olomouci, 2013.

Proof Index

corollary-1
corollary-1 (*prednaska/04*), 30

definition-0
definition-0 (*prednaska/07*), 49

definition-1
definition-1 (*prednaska/06*), 42

definition-11
definition-11 (*prednaska/02*), 21

definition-13
definition-13 (*prednaska/02*), 21

definition-2
definition-2 (*prednaska/06*), 42

definition-3
definition-3 (*prednaska/06*), 42

definition-4
definition-4 (*prednaska/06*), 42

definition-5
definition-5 (*prednaska/06*), 43

definition-6
definition-6 (*prednaska/02*), 17

definition-7
definition-7 (*prednaska/06*), 43

definition-8
definition-8 (*prednaska/06*), 43

example-4
example-4 (*prednaska/03*), 27

example-6
example-6 (*prednaska/01*), 8

example-7
example-7 (*prednaska/03*), 28

remark-1
remark-1 (*prednaska/07*), 50

remark-12
remark-12 (*prednaska/02*), 21

remark-14
remark-14 (*prednaska/02*), 21

remark-2
remark-2 (*prednaska/07*), 51

remark-3
remark-3 (*prednaska/03*), 27

remark-4
remark-4 (*prednaska/02*), 16

remark-5
remark-5 (*prednaska/03*), 28

remark-6
remark-6 (*prednaska/03*), 28

remark-7
remark-7 (*prednaska/02*), 17

remark-8
remark-8 (*prednaska/02*), 18

theorem-0
theorem-0 (*prednaska/04*), 29

theorem-10
theorem-10 (*prednaska/02*), 18

theorem-2
theorem-2 (*prednaska/03*), 24

theorem-3

theorem-3 (*prednaska/02*), 16

theorem-6

theorem-6 (*prednaska/06*), 43

theorem-9

theorem-9 (*prednaska/02*), 18