

MASARYKOVA UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA
ÚSTAV MATEMATIKY A STATISTIKY

Bakalářská práce

BRNO 2023

ŠTĚPÁN ZAPADLO

Dynamické systémy řízené daty

Bakalářská práce

Štěpán Zapadlo

Bibliografický záznam

Autor:	Štěpán Zapadlo Přírodovědecká fakulta, Masarykova univerzita Ústav matematiky a statistiky
Název práce:	Dynamické systémy řízené daty
Studijní program:	Matematika
Studijní obor:	Modelování a výpočty
Vedoucí práce:	doc. RNDr. Lenka Příbylová, Ph.D.
Akademický rok:	2022/2023
Počet stran:	xv + 91
Klíčová slova:	dynamické systémy; strojové učení; SINDy; matematická optimalizace; řídká regrese

Bibliographic Entry

Author:	Štěpán Zapadlo Faculty of Science, Masaryk University Department of Mathematics and Statistics
Title of Thesis:	Data-driven dynamical systems
Degree Programme:	Mathematics
Field of Study:	Modelling and computations
Supervisor:	doc. RNDr. Lenka Příbylová, Ph.D.
Academic Year:	2022/2023
Number of Pages:	xv + 91
Keywords:	dynamical systems; machine learning; SINDy; mathematical optimization; sparse regression

Abstrakt

V bakalářské práci se věnujeme identifikaci nelineární dynamiky systému na základě dat pocházejících z tohoto systému, jmenovitě pomocí metody řídké identifikace nelineární dynamiky (SINDy). První část práce uvádí potřebnou teorii z dynamických systému, lineárních regresních modelů, optimalizace a práce s šumem. Dále pak formulujeme SINDy metodu a uvádíme její implementaci v jazyce Julia, stejně jako časté nástrahy a problémy tohoto přístupu. Některé z nich se pokoušíme omezit vlastní optimalizační metodou *DSTLS* a uvádíme srovnání vybraných metod v závislosti na zašumění dat. V neposlední řadě podrobně analyzujeme simulační studie FitzHughova–Nagumova modelu a Lorenzova modelu. Nakonec se zaměříme na transformaci dynamiky Hodgkinova–Huxleyho modelu a FitzHughova–Nagumova spolu s optimalizační metodou pro nalezení obecné transformace mezi zmíněnými modely.

Abstract

In this thesis, we focus on the identification of nonlinear dynamics of a given system based on data from it, namely using the Sparse Identification of Nonlinear Dynamics (SINDy) framework. This thesis begins with the necessary theory of dynamical systems, linear regression models, optimization and noise handling. Then follows a mathematical formulation of the SINDy method, along with the implementation in Julia language and a discussion about common problems and challenges of this method. We attempt to address some of those issues by formulating our own optimization method *DSTLS* and provide a comparison of chosen methods in regard to noise levels in input data. Last but not least, we closely examine simulation studies of FitzHugh–Nagumo and Lorenz models. In the end, we focus on transformation of dynamics of the Hodgkin–Huxley model onto FitzHugh–Nagumo model, along with introducing an optimization framework for finding a general transformation between said models.

ZADÁNÍ
BAKALÁŘSKÉ PRÁCE

Akademický rok: 2022/2023

Ústav:	Ústav matematiky a statistiky
Student:	Štěpán Zapadlo
Program:	Matematika
Specializace:	Modelování a výpočty

Ředitel ústavu PřF MU Vám ve smyslu Studijního a zkušebního řádu MU určuje bakalářskou práci s názvem:

Název práce:	Dynamické systémy řízené daty
Název práce anglicky:	Data-driven dynamical systems
Jazyk závěrečné práce:	čeština

Oficiální zadání:

Cílem práce je prostudovat moderní trendy v oblasti dynamických systémů řízených daty, objevování dynamiky nelineárních systémů z dat pomocí lineární analýzy, zejména v souvislosti s modelováním neuronů.

Literatura: Brunton, Steven L., and J. Nathan Kutz. Data-driven science and engineering: Machine learning, dynamical systems, and control. Cambridge University Press, 2022. k dispozici na <http://databookuw.com/databook.pdf>

Vedoucí práce:	doc. RNDr. Lenka Příbylová, Ph.D.
Datum zadání práce:	5. 6. 2022
V Brně dne:	2. 4. 2023

Zadání bylo schváleno prostřednictvím IS MU.

Štěpán Zapadlo, 17. 10. 2022

doc. RNDr. Lenka Příbylová, Ph.D., 18. 10. 2022

RNDr. Jan Vondra, Ph.D., 26. 10. 2022

Poděkování

Na první místě bych chtěl poděkovat především doc. RNDr. Lence Příbylové, Ph.D. za odborné vedení, cenné rady, ale také ochotu, nadšení a podporu při psaní této práce. Dále bych chtěl poděkovat přítelkyni, rodině a kamarádům za trpělivost a podporu, kterou mi po celou dobu poskytovali.

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením vedoucí práce s využitím informačních zdrojů, které jsou v práci citovány.

Brno 9. května 2023

.....
Štěpán Zapadlo

Obsah

Přehled použitého značení	xv
Úvod	1
Kapitola 1. Základní pojmy	3
1.1 Julia	3
1.1.1 Multiple dispatch	3
1.1.2 Matematické výpočty	5
1.1.3 Cykly a iterace	9
1.1.4 Balíčky a grafy	11
1.2 Dynamické systémy	13
1.3 Lineární regrese	17
1.3.1 Metoda nejmenších čtverců	18
1.3.2 Řídká regrese	20
1.4 Odstraňování šumu	29
1.4.1 Klasifikace šumů	29
1.4.2 Metoda totální variace	32
1.4.3 Numerické derivování regularizované totální variací	33
Kapitola 2. Řídká identifikace nelineární dynamiky (SINDy)	35
2.1 Teoretický úvod	35
2.2 Ukázková implementace	37
2.3 Knihovna DataDrivenDiffEq.jl	38
2.4 Výzvy při hledání modelů	40
2.4.1 SINDy a data	41
2.4.2 Souřadnicový systém	43
2.4.3 Výběr členů do knihovny	44
2.4.4 Optimalizační algoritmy	45
Kapitola 3. Simulační studie	51
3.1 FitzHughův–Nagumův model	51
3.2 Lorenzův model	58
3.3 Transformace dynamiky Hodgkinova–Huxleyho modelu na FitzHughův–Nagumův model	67

Závěr	81
Přílohy	83
Seznam použité literatury	87

Přehled použitého značení

Pro snazší orientaci v textu zde čtenáři předkládáme přehled základního značení, které se v celé práci vyskytuje.

$\mathbb{R}, \mathbb{R}_{+\infty}$	množina všech reálných čísel, $\mathbb{R} \cup \{+\infty\}$
$\mathbb{N}, \mathbb{N}_0$	množina všech přirozených čísel, $\mathbb{N} \cup \{0\}$
$\ \cdot\ _k$	vektorová l_k -norma
$x, \mathbf{x}$	skalární, vektorová proměnná
$f, \mathbf{f}$	skalární, vektorová funkce (dle situace ¹)
$\mathcal{N}$	náhodná veličina, resp. vektor (dle situace)
$\text{sign}(x)$	signum (znaménko) proměnné x
$\dot{x} = \frac{d}{dt}x$	derivace funkce $x(t)$ podle času t
$\text{grad}_{\mathbf{x}} f = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right)$	gradient funkce f podle vektorové proměnné $\mathbf{x}$
$\nabla_{\mathbf{x}}^2 f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$	Hessova matice 2. derivací funkce f podle $\mathbf{x}$
$\text{prox}_f(\mathbf{v})$	proximální operátor funkce f , viz (1.10)
$\mathcal{D}(f)$	definiční obor funkce f
$\mathcal{L}_\rho(\mathbf{x}, \mathbf{u}) = f(\mathbf{x}) + \mathbf{u}^T \mathbf{h}(\mathbf{x}) + \frac{\rho}{2} \ \mathbf{h}(\mathbf{x})\ _2^2$	rozšířený lagrangian úlohy $\min_{\mathbf{x}} f(\mathbf{x})$ s podmínkou $\mathbf{h}(\mathbf{x}) = 0$
$\mathbf{X}$	matice plánu (nejčastěji matice měření stavových proměnných dynamického systému)
$\Theta(\mathbf{X})$	knihovna kandidátních funkcí aplikovaná na matici měření stavových proměnných
Ξ	matice koeficientů (nejčastěji) šetrného modelu

¹V této práci kombinujeme teorii dynamických systémů, zobecněnou lineární regresi a optimalizační metody. Zavedené značení v těchto oborech se ale navzájem překrývá, proto nelze jednoduše uvést použité značení.

Úvod

V dnešní době máme velké množství dat o mnoha různých systémech, avšak často neumíme vysvětlit jejich chování, ani ho ku příkladu predikovat. Ačkoliv nám určitý vhled do zkoumaného systému poskytnou data samotná, zvláště pro některé dynamické systémy můžeme chtít zjistit diferenciální rovnice, kterými se řídí.

V této práci se budeme věnovat problému identifikace nelineární dynamiky systému z dat, která o tomto systému máme. Na vstupu budeme mít časovou řadu nebo trajektorii ze zkoumaného systému a naším úkolem bude pomocí metod kombinujících lineární regresi, matematickou optimalizaci a dynamické systémy nalézt takový dynamický systém, který dobře vystihuje data a zároveň je např. i interpretovatelný. Využívat budeme metodu *řídce identifikace nelineární dynamiky* (SINDy) představenou v [1].

Text práce je členěn do 3 kapitol, přičemž úvodní kapitola čtenáře seznamuje s nutnými pojmy pro porozumění SINDy metodě. Nejprve nabízí úvod do programovacího jazyka **julia**, ve kterém jsou všechny algoritmy implementovány. Dále pak prochází základní pojmy dynamických systému a ilustruje je na názorných obrázcích. Poté se budeme věnovat metodám řídké lineární regrese, a to přesněji LASSO regresi řešené pomocí ADMM algoritmu a metodě *Sequentially Thresholded Least Squares* (STLS). V neposlední řadě se podíváme na typy šumu, které mohou ovlivnit vstupní data, a na metody, jak tento šum odstranit a jak numericky spočítat derivace ze zašuměných dat.

Druhá kapitola se věnuje samotné SINDy metodě. V první řadě se zaměříme na teoretickou formulaci SINDy metody za použití pojmů z první kapitoly. Dále pak uvedeme vlastní implementaci v jazyce **julia** a srovnáme ji s již stávající implementací z volně dostupného balíčku `DataDrivenDiffEq.jl`. Na konci této kapitoly se zaměříme na různé problémy, se kterými se můžeme při použití SINDy metody setkat, a na způsoby, jak je zmírnit, či dokonce odstranit. Zde také uvedeme srovnání různých optimalizačních metod v závislosti na velikosti šumu v datech a představíme námi navrhovanou optimalizační metodu *DSTLS*.

V poslední kapitole detailně prozkoumáme chování SINDy na 2 vybraných modelech. Nejdříve se zaměříme na FitzHughův–Nagumův model neuronu, ve kterém se objevují oscilace. Dále pak naši pozornost přesuneme k Lorenzově modelu, který vykazuje chaotické chování. U obou modelů popíšeme proces použití SINDy metody, řešení problémů, se kterými jsme se setkali, a srovnání výsledků pomocí různých přístupů. Poslední sekce této kapitoly je zaměřená na transformaci dynamiky Hodgkinova–Huxleyho modelu na jednodušší FitzHughův–

Nagumův model. Představíme nejen FitzHughovu původní lineární transformaci, ale i metodu, jak nalézt vhodné lineární i nelineární transformace, a poté ji ilustrujeme pro různé transformace na trajektorii Hodgkinova–Huxleyho modelu s tlumenými i netlumenými oscilacemi.

Práce je vysázena pomocí systému $\text{\LaTeX}$ a všechny originální obrázky jsou vytvořeny pomocí jazyka `julia` a balíčku `Makie.jl`. Veškerý použitý kód, dokumentaci, obrázky a další soubory lze najít v [GitLab repozitáři](#)². V tomto repozitáři se také nachází návod na spuštění vytvořeného kódu lokálně.

²Tento repozitář je dostupný na adrese <https://gitlab.com/sceptri/bachelor-thesis>.

Kapitola 1

Základní pojmy

1.1 Julia

Tato práce se zabývá algoritmy strojového učení, které se používají při identifikaci a nalézání dynamiky nelineárních dynamických systémů (viz podkapitola 1.2). Ukázkové implementace jsou uvedeny v jazyku **julia**, proto se sluší uvést její syntax a jak **julia** funguje.

Jazyk **julia** je dynamicky typovaný vysokoúrovňový programovací jazyk, který byl navržen primárně pro vědecké účely, viz [2]. Její vývoj započal v roce 2009 a v roce 2012 byla vydána jako „open-source“. Kompilace **julia** kódu probíhá tzv. *JIT* (z anglického *Just-In-Time*), tedy při prvním volání libovolné funkce se musí tato funkce zkompileovat. To sice znamená, že napoprvé je všechno pomalé, ale opakovaná volání pak stojí zlomek času.

Zastáváme názor, že **julia** představuje další krok ve vědeckých výpočtech nejen svou syntaxí, ale i bohatou základnou uživatelů a dostupných balíčků či také rychlostí¹. Jakožto open-source projekt inspiruje dále pokračovat a posunovat „open-science“. Myslíme si, že **julia** usnadňuje nejen vývoj nových balíčků, ale i jejich pochopení dalšími zájemci – v ostatních jazycích (MATLAB, , Python), používaných pro vědecké účely, jsou kritické komponenty balíčků (či jazyka samotného) napsány v jiném nízkoúrovňovém jazyku, aby byl výsledný produkt dostatečně rychlý². Naopak většina **julia** balíčků a knihoven je napsaná v ní samotné. Proto se domníváme, že si **julia** zaslouží delší a detailnější představení, než které by bylo potřeba pro ostatní jazyky.³

1.1.1 Multiple dispatch

V celku nestandardně **julia** nevyužívá principy objektově orientovaného programování (zkráceně *OOP*), ale místo toho implementuje paradigma *multiple dispatch*, což lze volně přeložit jako *vícenásobné volání*. Při tomto přístupu může mít funkce jednoho jména více implementací, přičemž při zavolání této funkce se vybere ta

¹Srovnání oproti ostatním jazykům [zde](#).

²Toto zejména platí o knihovnách pro strojové učení.

³Celá tato sekce je inspirovaná výukovými materiály [3].

nejvhodnější podle počtu argumentů a jejich typů.

Ukažme si *multiple dispatch* na jednoduchém příkladu. Mějme funkci `hello` se třemi implementacemi (v řeči `julia` jsou to tzv. metody):

```
function hello()
    return "Hello!"
end
function hello(name)
    return "Oh hi, $name";
end
function hello(number::Number)
    return "Hello $number times";
end
```

a potom

```
julia> hello()
"Hello!"
julia> hello("John")
"Oh hi, John"
julia> hello(3)
"Hello 3 times"
```

Tento příklad pěkně ilustruje i syntax `julia` při psaní funkcí, přičemž ale můžeme použít i kratší zápis

```
julia> hello(char) = string("Hello to ", char)
hello (generic function with 3 methods)
julia> hello('u')
"Hello to u"
```

Argumenty funkcí samozřejmě mohou mít nastavené výchozí hodnoty, např. `hello(name = "John")`, které se použijí, pokud při volání funkce nedodáme hodnotu argumentu explicitně. Avšak v případě, že máme funkci s velkým počtem argumentů (např. funkce vykreslující grafy), tak by bylo náročné nastavovat „pozdější“ argumenty⁴. Proto `julia` zavádí i tzv. „keyword arguments“ (klíčové argumenty, zkráceně „*kwargs*“), které si ukážeme na následujícím příkladu:

```
function hello(name; # kwargs should be separated by a semicolon
    greetings = "Hello", # kwargs MUST have a default value
    ending_char = "!"
)
    return "$greetings to $name$ending_char"
end
```

a potom

⁴Museli bychom ručně přepsat spoustu výchozích hodnot předchozích argumentů.

```
julia> hello("John")
"Hello to John!"
julia> hello("John"; ending_char = ".")
"Hello to John."
```

1.1.2 Matematické výpočty

Při programování (nejen) matematických modelů, a obecně ve vědeckém programování, je nutné zapisovat velké množství matematických rovnic či formulí. V této oblasti je syntax velmi podobná například MATLABu a Fortranu. Dále uvedme několik příkladů ilustrujících matematické výpočty v **julia**:

```
julia> a = [1, 2, 3, 4] # Defining a (column) vector
4-element Vector{Int64}:
 1
 2
 3
 4
julia> b = [1 2; 3 4] # Defining a 2 by 2 matrix
2×2 Matrix{Int64}:
 1  2
 3  4
julia> c = [1 2 3 4] # Or 1 by 4 matrix
1×4 Matrix{Int64}:
 1  2  3  4
```

Na předchozích příkladech lze vidět, jak je možné definovat vektory a matice. Nicméně při počítání budeme často potřeba i transpozici, kterou dostaneme v **julia** příkazem `c'`⁵. Jak jsme mohli vidět výše, když jsme definovali vektor a matici, přiřadila jim **julia** jiné typy, např.:

```
julia> c'
4×1 adjoint(::Matrix{Int64}) with eltype Int64:
 1
 2
 3
 4
julia> typeof(c), typeof(c')
(Matrix{Int64}, LinearAlgebra.Adjoint{Int64, Matrix{Int64}})
julia> typeof(c) <: AbstractMatrix, typeof(c') <: AbstractMatrix
(true, true)
julia> a'
1×4 adjoint(::Vector{Int64}) with eltype Int64:
 1  2  3  4
```

⁵Ve skutečnosti funkce `c'` vrací *komplexně konjugovanou* matici. Chceme-li přímo transponovanou matici, použijeme funkci `transpose`.

```
julia> typeof(a), typeof(a')
(Vector{Int64}, LinearAlgebra.Adjoint{Int64, Vector{Int64}})
julia> typeof(a) <: AbstractMatrix, typeof(a') <: AbstractMatrix
(false, true)
```

Funkce `typeof` nám vrací typ proměnné a pomocí operátoru `<:` jej můžeme „porovnat“ s nějakým jiným typem a zjistit, zda je typ na levé straně operátoru *podtypem* typu na pravé straně. Jelikož jsou operace s maticemi v **julia** definovány pro podtypy abstraktního⁶ typu `AbstractMatrix`, tak i maticové násobení jistě budeme moci provádět na podtypech `AbstractMatrix`, ale bez další implementace nikoliv s proměnnou typu `Vector`.

```
julia> a * a
Error: MethodError: no method matching *(::Vector{Int64},
  ↳ ::Vector{Int64})
Closest candidates are:
  *(::Any, ::Any, !Matched::Any, !Matched::Any...) at
  ↳ operators.jl:591
  *(!Matched::StridedMatrix{T}, ::StridedVector{S}) where
  ↳ {T<:Union{Float32, Float64, ComplexF32, ComplexF64}, S<:Real}
  ↳ at ~/packages/julias/julia-
  ↳ -1.8/share/julia/stdlib/v1.8/LinearAlgebra/src/matmul.jl:49
  *(::StridedVecOrMat, !Matched::LinearAlgebra.Adjoint{<:Any,
  ↳ <:LinearAlgebra.LQPackedQ}) at ~/packages/julias/julia-
  ↳ -1.8/share/julia/stdlib/v1.8/LinearAlgebra/src/lq.jl:269
  ...
julia> a * a' # ' returns a complex conjugate
4×4 Matrix{Int64}:
 1  2  3  4
 2  4  6  8
 3  6  9 12
 4  8 12 16
julia> c * c
Error: DimensionMismatch: matrix A has dimensions (1,4), matrix B
  ↳ has dimensions (1,4)
julia> c' * c
4×4 Matrix{Int64}:
 1  2  3  4
 2  4  6  8
 3  6  9 12
 4  8 12 16
```

⁶Abstraktní typy v **julia** slouží pro vystižení obecných případů typů proměnných, např. abstraktní typ `Number` je „nadpojem“ specifitějšího číselného datového typu `Integer`. Nicméně nemůžeme mít proměnnou pouze typu `Number` – každá proměnná je vždy nějakého konkrétního typu.

Všimněme si různých chybových hlášek při volání $a * a$ a $c * c$. Proměnná c je matice tvaru 1×4 , a tedy $c * c$ jednoduše není (*matematicky*) definované, neboť nesouhlasí rozměry matic. Naopak chybová hláška při volání $a * a$ nám říká, že **julia** nemá definované násobení 2 vektorů mezi sebou, jak již bylo zmíněno výše.

Další užitečnou operací, kterou budeme potřebovat s maticemi provádět, je inverze, která má v **julia** jméno `inv`, např.:

```
julia> b
2×2 Matrix{Int64}:
 1  2
 3  4
julia> inv(b)
2×2 Matrix{Float64}:
-2.0  1.0
 1.5 -0.5
```

Když potom řešíme regulární systém lineárních rovnic $Ax = b$, jehož řešení je $A^{-1}b$, tak buď můžeme použít `inv(A) * b`, nebo v **julia** existuje efektivnější způsob použitím *inverzního dělení*, tj. počítáme `A \ b`.

```
julia> A = [1 2; 3 4]; b = [1; 2];
julia> inv(A) * b
2-element Vector{Float64}:
 0.0
 0.5
julia> A \ b
2-element Vector{Float64}:
 0.0
 0.5
```

Pro znázornění rozdílů mezi těmito 2 přístupy použijme makro⁷ `@time`, které nám umožňuje měřit trvání jednotlivých příkazů a počet jejich alokací:

```
julia> using LinearAlgebra, Random # we need the norm function
└ from LinearAlgebra package and seed! function from Random
julia> Random.seed!(12345); # for reproducibility, we set a custom
└ RNG seed
julia> n = 1000; A = randn(n,n); b = randn(n); # create a random
└ n*n matrix and a n-dimensional vector
julia> norm(inv(A)*b - A \ b) # check that they both result in the
└ same matrix
1.1793595415142444e-12
julia> @time (inv(A) * b);
0.061986 seconds (6 allocations: 8.133 MiB, 18.35% gc time)
julia> @time (A \ b);
0.015699 seconds (4 allocations: 7.645 MiB)
```

⁷Makro je speciální „funkce“ používaná pro metaprogramování, která umožňuje generování kódu. Jinak řečeno, makro je vykonáno ještě předtím, než je zkompileován zbytek kódu.

Zde si všimněme, že skutečně byl výpočet přes $A \setminus b$ zhruba o 0.04s rychlejší na matici 1000×1000 . To se sice může zdát nevýznamné, ale později uvidíme, že uvedené metody pro nalezení rovnic z dat budou potřebovat relativně velké množství takovýchto operací na výrazně větších maticích. A proč je tedy jedna metoda rychlejší než druhá? Podívejme se na to, co se přesně děje při zavolání $A \setminus b$. V dokumentaci [4, 5] můžeme zjistit, že při zavolání $A \setminus b$ se ve skutečnosti vykoná výraz `qr(A, Val{true}()) \ b`. To znamená, že první uděláme *permutovaný* QR rozklad⁸ matice⁹ A a potom znovu voláme inverzní dělení. Povšimněme si ale, že z QR rozkladu **nedostáváme** podtyp `AbstractMatrix`, ale podtyp `Factorization`:

```
julia> A = [1 2; 3 4];
julia> qr_type = typeof(qr(A, Val{true}()))
LinearAlgebra.QRPivoted{Float64, Matrix{Float64}, Vector{Float64},
  ~ Vector{Int64}}
julia> qr_type <: AbstractMatrix, qr_type <: Factorization
(false, true)
```

Při inverzním dělení s faktorizací se `julia` dále snaží minimalizovat velikost residua¹⁰, neboť počítání $Ax = b$ je *de facto* stejné jako optimalizační problém $\min_x \|Ax - b\|_2^2$. QR rozklad matice A označme $A = QR$, kde Q je *ortogonální* matice rotace tvaru $n \times n$ a R je *horní trojúhelníková* matice tvaru $n \times p$ (předpokládáme, že A je tvaru $n \times p$). Je-li $n > p$, pak dokonce můžeme použít tzv. *ekonomický QR rozklad*, neboť matice R je jistě tvaru

$$R = \begin{pmatrix} R_1 \\ \mathbf{0} \end{pmatrix},$$

kde matice R_1 je horní trojúhelníková, a tedy při vynechání nulových řádků a jim odpovídajících sloupců z Q (zde můžeme Q rozložit na $Q = (Q_1 \ Q_2)$, kde Q_2 jsou sloupce odpovídající nulovým řádkům z R) dostáváme QR rozklad jako $A = Q_1 R_1$. Celkem pro chybu $\|Ax - b\|_2^2$ můžeme psát:

$$\begin{aligned} \|Ax - b\|_2^2 &= \|QRx - b\|_2^2 \\ &= \|Q^T(QRx - b)\|_2^2 \\ &= \left\| \begin{pmatrix} R_1 \\ \mathbf{0} \end{pmatrix} x - \begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix} b \right\|_2^2 \\ &= \|R_1 x - Q_1^T b\|_2^2 + \|Q_2^T b\|_2^2 \end{aligned}$$

Abychom dosáhli optima, je potřeba minimalizovat výraz $\|R_1 x - Q_1^T b\|_2^2$. Jeli-kož je matice R horní trojúhelníková, tak z konstrukce matice R_1 dostáváme, že

⁸Výpočet QR rozkladu se provádí pomocí [Householderových reflexí](#).

⁹Anglicky „*pivoted QR decomposition*“.

¹⁰Pouze pro obdélníkové matice A .

i tato matice je horní trojúhelníková. Pokud je matice A *plně hodnosti*, tak dokonce je matice R_1 i invertibilní a navíc výpočet její inverze nevyžaduje znalost determinantu. Výsledný vektor $\hat{x}$ pak dostáváme jako $\hat{x} = R_1^{-1}Q_1^T \mathbf{b}$.

Všimněme si, že na výpočet optimálního $\hat{x}$ jsme *nepotřebovali* spočítat (*drahou*) inverzi matice A . Naopak pokud bychom počítali $\text{inv}(A) * \mathbf{b}$, tak se první spočítá $\text{inv}(A) = \text{one}(A) / A^{11}$, dále se maticové dělení $\text{one}(A) / A$ převede na inverzní dělení $(A' \setminus (\text{one}(A))')'$ a až potom výsledek maticově vynásobíme s vektorem $\mathbf{b}$. Další nevýhodou tohoto postupu je, že si v paměti potřebujeme uchovat celou inverzi matice A , což v prvním případě nebylo nutné.

1.1.3 Cykly a iterace

V předchozí sekci jsme se mimo jiné zabývali maticovým násobením, což je velmi rychlý způsob, jak například spočítat součin dvou vektorů:

```
julia> a = [1, 2, 3]; b = [4, 5, 6];
julia> a' * b
32
```

Pokud bychom si rozepsali nutné operace, tak zjistíme, že nejprve po složkách vynásobíme vektory $\mathbf{a}$, $\mathbf{b}$ a pak tyto součiny sečteme. To můžeme zapsat v **julia** jako operaci po složkách, která se značí $.$ a potom operace, kterou chceme provést, tj. $.*$ pro násobení po složkách. Výsledek tohoto násobení nakonec sečteme:

```
julia> a .* b
3-element Vector{Int64}:
 4
 10
 18
julia> sum(a .* b)
32
```

Pokud bychom potřebovali více operací provést po složkách, můžeme dokonce použít makro `@.`, které z každé operace udělá operaci po složkách:

```
julia> c = [7, 8, 9];
julia> sum(a .* b .+ c)
56
julia> sum(@. a * b + c)
56
```

V praxi při programování zjistíme, že není vždy intuitivní používat maticové násobení a často je zápis pomocí cyklu výrazně čitelnější a přehlednější. V **julia** navíc neplatí, že by cykly byly pomalejší než maticové násobení, jako třeba v MATLABu – někdy může být správně zapsaný cyklus rychlejší, než vektori-

¹¹Funkce `one(x)` vrací jednotku pro typ daný x , tedy v případě matice vrací jednotkovou matici.

zace¹² daného problému.

V **julia** máme standardně 2 druhy cyklů – **for** a **while**, přičemž **for** se v mnohém chová jako **foreach** cyklus z jiných programovacích jazyků. Ukažme si je na příkladu:

```
julia> i = 1
1
julia> while i ≤ 5 # using while
    global i # we need to make sure that i is in scope
    print(i)
    i += 1
end
12345
julia> for i in 1:5 # using for
    print(i)
end
12345
```

V kódu výše jsme na každé číslo 1 až 5 aplikovali funkci **print**, která nám jej vytiskla na obrazovku. Tento koncept můžeme rozvést dále, tj. na každý prvek z vektoru budeme chtít aplikovat nějakou funkci. To bychom sice mohli řešit pomocí některého z výše uvedených způsobů, ale **julia** nám ještě nabízí jednu možnost pomocí tzv. *list comprehension* nebo generátorů:

```
julia> a = [x^2 for x in 1:5] # list comprehension
5-element Vector{Int64}:
 1
 4
 9
16
25
julia> b = (x^2 for x in 1:5) # generator syntax
Base.Generator{UnitRange{Int64},
  ↳ Main.WeaveSandBox0.var"#4#5"}(Main.WeaveSandBox0.var"#4#5"(),
  ↳ 1:5)
julia> collect(b)
5-element Vector{Int64}:
 1
 4
 9
16
25
```

Použití generátorů je v **julia** relativně časté, protože nám umožňuje pracovat s velkými vektory, aniž bychom je potřebovali celé uchovávat v paměti. V ukáz-

¹²Vektorizace je proces, při kterém problém dobře řešitelný cyklem zapíšeme jako maticové násobení.

kách jsme vždy iterovali přes nějaké pole, vektor nebo generátor¹³, nicméně můžeme chtít iterovat i přes námi definovaný typ. V takovém případě potřebujeme předefinovat funkci `iterate` z hlavního balíčku `Base`:

```
# First we create a new type named forIteration,
# which has a field array and a default constructor
struct forIteration{T<:AbstractArray}
    array::T
end

Base.length(object::forIteration) = length(object.array)
Base.iterate(object::forIteration; kwargs...) =
    ↪ Base.iterate(object, 1; kwargs...)

function Base.iterate(object::forIteration, index; kwargs...)
    if index > length(object)
        return nothing; # by returning nothing we end the
    ↪ iteration process
    end
    current_element = object.array[index] # take the index-th
    ↪ element of the object.array
    return (current_element, index + 1)
end
```

Syntaxe `struct` nám umožňuje vytvořit si vlastní typ, zde pojmenovaný `forIteration`, který má námi definované atributy – v tomto případě pouze atribut `array`, který je podtypem abstraktního typu `AbstractArray`. Dále bychom mohli ještě definovat potřebné konstruktory¹⁴, nicméně tentokrát si vystačíme s výchozím konstruktorem. Poté můžeme už jen použít:

```
julia> custom = forIteration([1,2,3])
Main.WeaveSandBox0.forIteration([1, 2, 3])
julia> [x^2 for x in custom]
3-element Vector{Int64}:
 1
 4
 9
```

1.1.4 Balíčky a grafy

Stejně jako například v , tak i v  můžeme používat balíčky na získání nových funkcí a typů pro cokoliv, co můžeme k práci potřebovat. Obecně se k správci balíčků dostaneme z  konzole pomocí `]` a potom pro nainstalování balíčku použijeme příkaz `add`, např. `add Random` nainstaluje balíček `Random`. To stejné můžeme udělat čistě v kódu pomocí balíčku `Pkg`, viz následující příklad:

¹³Syntax `1:10` ve skutečnosti vrací generátor, nikoliv vektor, čísel od 1 do 10.

¹⁴Funkce, které určují, jak můžeme inicializovat daný typ.

```
using Pkg
Pkg.add("Random")
```

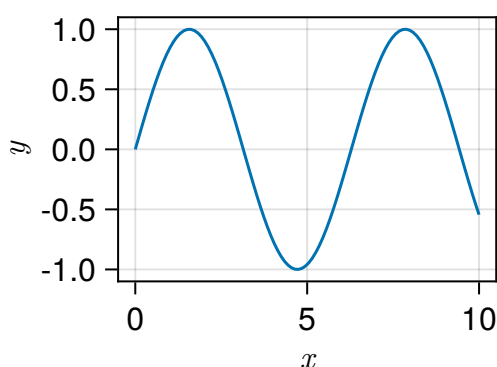
Jak je vidět na tomto příkladu, pomocí `using PackageName` načteme všechny *exportované* funkce a další symboly z daného balíčku do našeho vývojového prostředí. Budeme-li chtít přistoupit k nějakému neexportovanému symbolu¹⁵, použijeme syntax `PackageName.funcName`, což jsme mohli vidět výše na `Pkg.add`.

Při spuštění **julia** se dostaneme do *globálního vývojového prostředí*, nicméně když pracujeme na nějakém projektu, je dobré změnit prostředí pouze na tuto složku. To uděláme pomocí příkazu `] activate .`, což nám aktivuje prostředí v momentálním adresáři. Pokud poté nainstalujeme nějaký balíček, vytvoří se nám v adresáři soubory `Project.toml` a `Manifest.toml`, které určují, jaké balíčky mají být v tomto vývojovém prostředí instalovány a v jaké verzi.

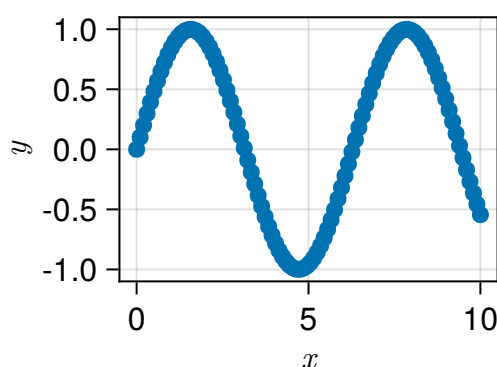
Na vytváření grafů existuje v **julia** spousta balíčků, od základního `Plots` [6], přes `Makie` a `Gadfly`, až po `Vega`. V této práci budeme používat balíček `Makie` [7]. Ukažme si jednoduchý příklad:

```
using CairoMakie, LaTeXStrings
```

```
resolution = (280, 210) # Define needed resolution 280pt x 210pt
f = Figure(; resolution) # Create a figure with a given resolution
ax = Axis(f[1, 1], xlabel=L"x", ylabel=L"y") # Add 1 axis to the
    ↪ figure, with axis labels x and y written in LaTeX
x = range(0, 10, length=100)
y = sin.(x)
lines!(ax, x, y) # Connect (x,y) coordinates by lines
```



(a) Ukázka použití funkce `lines!`



(b) Ukázka použití funkce `scatter!`

Obrázek 1.1: Ukázka použití knihovny `Makie`

V předcházejícím kódu (viz 1.1a) jsme si prvně vytvořili nové „plátno“ pomocí `Figure` a definovali jsme si rozměr plátna na 280 pt × 210 pt. Dále jsme určili, že

¹⁵Nebo pokud jsme použili příkaz `import`, který nepřidává symboly do vývojového prostředí.

na plátně má být 1 osa, vytvořená pomocí `Axis`, s popisy os x a y , do které jsme vykreslili sinus jako úsečky spojující body pomocí `lines!(ax, x, y)`. Podobně bychom mohli použít např. `scatter!` a dostali bychom graf znázorňující pouze body (viz 1.1b).

Další balíčky, které budeme potřebovat, uvedeme při jejich prvním použití.

1.2 Dynamické systémy

Dynamickým systémem nazýváme model, jehož stav se mění v čase, přičemž tuto změnu popisujeme matematickými rovnicemi. Může se jednat například o fyzikální, chemický, ekologický či ekonomický model – příkladem stavu systému pak může být poloha částic v prostoru nebo počet živočichů daného druhu v dané lokalitě. V této sekci vycházíme hlavně z [8] a [9]. Na simulace a řešení diferenciálních rovnic budeme používat balíček `DifferentialEquations.jl` [10], případně i `ModelingToolkit.jl` [11].

Definice 1. Dynamický systém rozumíme trojici $\{T, X, \varphi^t\}$, kde číselná množina T je monoid (čas), X je metrický prostor, který nazýváme **fázovým prostorem**, a φ^t je parametrický systém evolučních operátorů s parametrem $t \in T$ definovaných jako zobrazení $\varphi^t : X \rightarrow X$, které zobrazuje počáteční stav $x_0 \in X$ na nějaký stav $x_t = \varphi^t x_0 \in X$.

V případě, že $T = \mathbb{Z}$, pak hovoříme o *diskrétním dynamickém systému*, naopak pro $T = \mathbb{R}$ jde o *spojitý dynamický systém*. Je-li přechod z jednoho stavu do druhého vyjádřen pomocí pravděpodobností, pak takový systém nazýváme *stochastický* – v opačném případě jde o **deterministický** systém, kterým se budeme výhradně věnovat.

Definice 2. Deterministický dynamický systém nazveme systém právě tehdy, když trojice $\{T, X, \varphi^t\}$ splňuje podmínku

$$\varphi^0 = \text{id},$$

kde id je identita na X , tj. $\forall x \in X : \text{id } x = x$.

Jsou-li zákony daného systému v čase neměnné, pak jde tzv. *autonomní systém*. V praxi může jít například o fyzikální zákony, či vztahy popisující vývoj populace v uzavřeném prostředí.

Definice 3. Autonomní dynamický systém rozumíme deterministický systém $\{T, X, \varphi^t\}$ splňující podmínku

$$\varphi^{t+s} = \varphi^t \circ \varphi^s,$$

tj. $\forall x \in X : \varphi^{t+s} x = \varphi^t(\varphi^s x)$, jsou-li definovány obě strany rovnice.

Jakožto příklad spojitého autonomního systému můžeme uvést systém s jed-

nou stavovou proměnou $x(t)$, která závisí na čase, přičemž změna této proměnné je dána *obyčejnou diferenciální rovnicí*

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

kde $t \in \mathbb{R}$ (často také pouze $t \in \mathbb{R}^+$) a $f : \mathbb{R} \rightarrow \mathbb{R}$. Dále příkladem diskrétního systému může systém podobný tomu uvedenému výše, avšak u kterého bude změna stavu x dána pomocí diferenční rovnice

$$x_{n+1} = f(x_n),$$

kde $n \in \mathbb{N}$ (případně $n \in \mathbb{Z}$) a $f : \mathbb{R} \rightarrow \mathbb{R}$.

Je dobré si uvědomit, že *spojitý autonomní systém* zadaný obyčejnou diferenciální rovnicí (ODR) vyššího řádu můžeme upravit na *spojitý autonomní systém* zadaný systémem ODR prvního řádu.

V praxi pomocí modelů zkoumáme reálné situace, např. jak se vyvíjí populace v ekosystému v čase nebo jakou rychlost má v daném časovém okamžiku kyvadlo – jinak řečeno se díváme na to, jak se mění stav systému, když postupuje časem.

Definice 4. Trajektorie s počátečním bodem $\mathbf{x}_0 \in \mathbf{X}$ je uspořádaná podmnožina fázového prostoru $\mathbf{X}$

$$\{\mathbf{x} \in \mathbf{X} \mid \mathbf{x} = \boldsymbol{\varphi}^t \mathbf{x}_0, \forall t \in T, \text{ pro které je } \boldsymbol{\varphi}^t \mathbf{x}_0 \text{ definováno}\}.$$

V případě spojitého systému jde o orientované křivky v $\mathbf{X}$, v případě diskrétního systému jsou to posloupnosti bodů v $\mathbf{X}$. **Fázovým portrétem** dynamického systému rozumíme rozmístění trajektorií ve fázovém prostoru $\mathbf{X}$.

Trajektorie často zobrazujeme jako funkce (resp. posloupnosti) času a počátečního bodu, tj. zobrazení $T \times \mathbf{X} \rightarrow \mathbf{X}$. První si ukažme příklad fázového portréту pro jednorozměrný systém se stavovou proměnou $x(t)$ reprezentující populaci nějakého druhu v uzavřeném prostředí (jde o tzv. [Verhulstův model růstu populace](#))

$$\dot{x}(t) = x(t) \cdot r_0 \cdot \underbrace{\left(1 - \frac{x(t)}{K}\right)}_{r(x(t))},$$

kde r_0 je *růstový koeficient* a K kapacita prostředí. Pomocí následujícího kódu můžeme dostat fázový portrét 1.2 našeho systému.

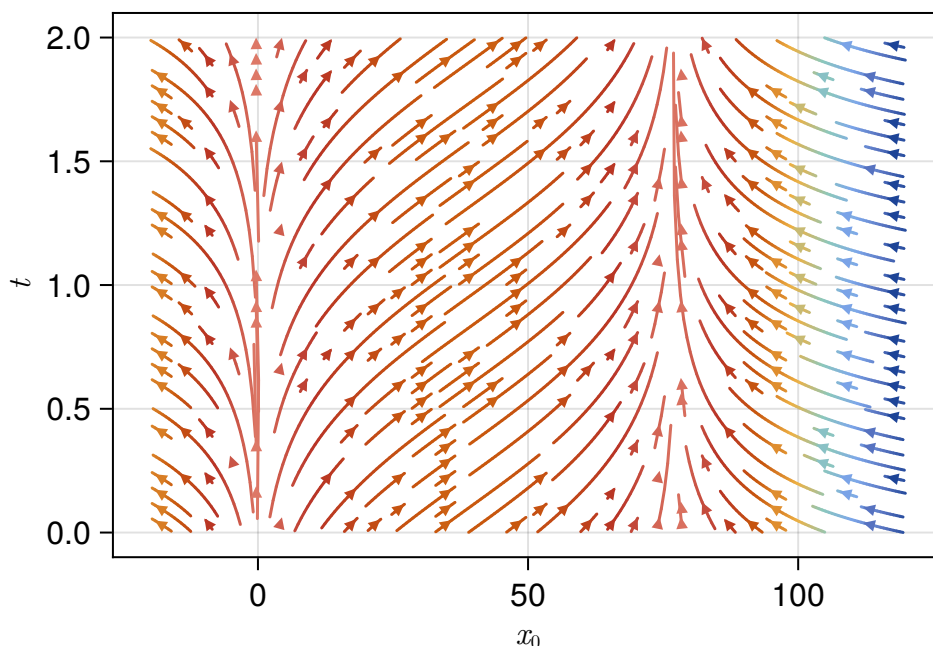
```

r0 = 3.3; K = 77
ẋ(x) = @. x * r0 * (1 - x / K) # Define the equation governing the
    ↪ model

fig = Figure(; resolution)
ax = Axis(fig[1, 1]; labels...)

let f(x,t) = Point2f(ẋ(x), 1)
    streamplot!(ax, f, -20..120, 0..2; params...)
end

```



Obrázek 1.2: Fázový portrét jednorozměrného spojitého autonomního modelu

Šipky na fázovém portréту 1.2 ukazují, jak se vyvíjí populace v čase – vektorové pole „teče“ stejným směrem, kterým šipky ukazují.

Všimněme si, že na ose pro x_0 existují dva významné body – bod vymření populace $x_0 = 0$ a bod kapacity prostředí $x_0 = K = 77$. Jakmile se systém dostane do jednoho z těchto dvou bodů, tak už v něm zůstane, proto se takové body nazývají *stacionární*.

Definice 5. Bod $\mathbf{x}_0 \in \mathbf{X}$ nazýváme **stacionárním bodem** (nebo též *rovnovážným, singulárním, či pevným* bodem) dynamického systému, jestliže pro všechna $t \in T$ platí

$$\varphi^t \mathbf{x}_0 = \mathbf{x}_0.$$

Ačkoliv je fázový portrét pěkná a užitečná vizualizace i pro jednorozměrné systémy, jeho předností je zobrazování dvourozměrných systémů. Vezměme si ku příkladu FitzHughův–Nagumův model neuronu, který budeme průběžně zkoumat i ve zbytku práce. Jedná se o *autonomní systém diferenciálních rovnic*.

Definice 6. Autonomním systémem diferenciálních rovnic rozumíme systém

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}), \tag{1.1}$$

kde $\mathbf{x} \in \mathbf{X} = \mathbb{R}^n$ a vektorová funkce $\mathbf{f} : \mathbb{R}^m \rightarrow \mathbb{R}^n$ je *dostatečně hladká*.¹⁶

Poznámka. Rovnovážné body autonomního systému (1.1) splňují systém rovnic¹⁷

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}.$$

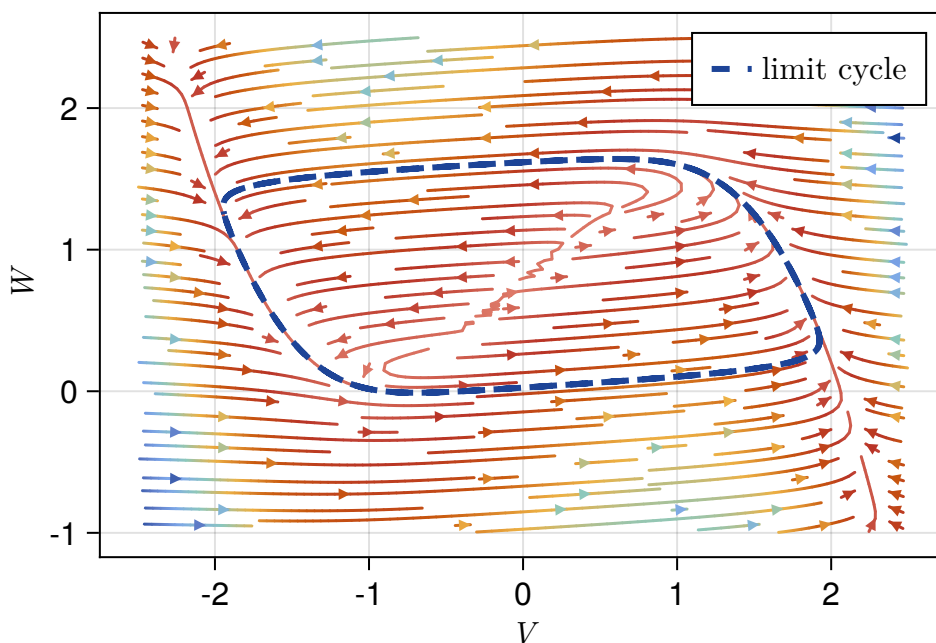
¹⁶Obdobně můžeme zavést *autonomní systém diferenčních rovnic*, viz [8].

¹⁷Pro stacionární body autonomního systému diferenčních rovnic platí $\mathbf{f}(\mathbf{x}) = \mathbf{x}$.

FitzHughův–Nagumův model je dvourozměrný systém definovaný vztahy

$$\begin{aligned}\dot{V}(t) &= V(t) - \frac{V^3(t)}{3} - W(t) + i_e, \\ \dot{W}(t) &= a \cdot (b \cdot V(t) - c \cdot W(t) + d),\end{aligned}\tag{1.2}$$

kde V odpovídá **membránovému potenciálu**, W je *obnovovací proměnná*, i_e velikost budícího proudu¹⁸ a a, b, c, d jsou parametry, viz sekce 3.1. Jedná se o zjednodušení 4-rozměrného Hodgkinova–Huxleyho modelu, kterému se také budeme věnovat později v sekci 3.3.



Obrázek 1.3: Fázový portrét FitzHughova–Nagumova modelu pro $i_e = 0.8$ ([zdrojový kód](#))

Šipky v obrázku 1.3 opět znázorňují plynutí času. Zde si navíc povšimněme uzavřené trajektorie (značené čárkovaně), ke které jsou ostatní trajektorie přitahovány. Tato trajektorie tvoří tzv. *invariantní množinu* – jakmile se nacházíme v této množině, už se z ní nedostaneme – přesněji je to tzv. *limitní cyklus*.

Definice 7. Invariantní množinou S rozumíme podmnožinu X splňující

$$x_0 \in S \implies \varphi^t x_0 \in S \quad \forall t \in T.$$

Definice 8. Uzavřená invariantní množina S se nazývá **stabilní**, jestliže

- pro každé libovolné malé okolí U množiny S existuje okolí V množiny S takové, že $\forall x \in V$ a $\forall t > 0$ platí $\varphi^t x \in U$ (tento typ stability nazýváme **Ljapunovskou stabilitou**),

¹⁸Více informací [zde](#) nebo [8, str. 269].

- existuje okolí U_0 množiny S takové, že $\varphi^t x \rightarrow S$ pro $x \in U_0$ a $t \rightarrow \infty$ (tento typ stability nazýváme **asymptotickou stabilitou**).

Množina S se též nazývá **atraktorem**.

Definice 9. Cyklem rozumíme periodickou trajektorii L , která není rovnovážným bodem, splňující $\forall x_0 \in L$

$$\varphi^{t+T_0}x_0 = \varphi^t x_0,$$

pro nějaké $T_0 > 0, \forall t \in T$. Nejmenší takové T_0 nazýváme *periodou cyklu L* . **Limitním cyklem** rozumíme cyklus, v jehož okolí nejsou jiné cykly.

Poznámka. V systému s cyklem vznikají periodické oscilace. Cyklus spojitého systému je *uzavřená křivka* v X , naopak cyklus diskrétního systému je *konečná uspořádaná k -tice bodů* z X .

V řeči výše uvedených definic tedy můžeme tvrdit, že se pro námi zvolené parametry v systému daném FitzHughovým–Nagumovým modelem vyskytuje *stabilní limitní cyklus*. Pokud se vrátíme k Verhulstovu modelu, tak můžeme říci, že se v něm (pro zobrazené parametry) nachází stabilní (*přitahující, atrahující*) a nestabilní (*odpuzující*) rovnovážný bod. Na závěr si uveďme větu popisující typy trajektorií, se kterými se lze setkat.

Věta 1.1. *Libovolná trajektorie řešení autonomního systému (1.1) je právě jednoho z typů:*

- **stacionární bod** (odpovídá konstantnímu řešení),
- **cyklus** (odpovídá nekonstantnímu periodickému řešení),
- *trajektorie, která sama sebe neprotíná.*

Důkaz. Viz [9, str. 82]. □

1.3 Lineární regrese

Tato práce nese název „Dynamické systémy řízené daty“. Z toho můžeme vytušit, že na začátku celého procesu nalézání rovnic budeme mít nějaká data, například měření stavových proměnných $x_1, \dots, x_k$, ze kterých poté budeme vyvozovat hodnoty závislých proměnných. Nejčastěji budeme konstruovat lineární model, který obecně má tvar

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_Y = \underbrace{\begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,k} \\ x_{2,1} & x_{2,2} & \dots & x_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \dots & x_{n,k} \end{bmatrix}}_X \underbrace{\begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_k \end{bmatrix}}_{\xi} + \underbrace{\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}}_{\varepsilon}, \quad (1.3)$$

kde $x_{i,j}$ jsou naměřené hodnoty (**kovariáty, prediktory**), ξ_i jsou **regresní koeficienty**, y_i je **závislá proměnná (odezva)** a ε_i je (*náhodná*) chyba pozorování.

Dále pro naše potřeby upravíme rovnici (1.3) tak, že pro $i = 1, \dots, n$ nebudeme predikovat pouze skalární proměnnou y_i , ale vektor $\mathbf{y}_i = [y_{i,1} \ y_{i,2} \ \dots \ y_{i,l}]$. Navíc nám většinou nebudou stačit čistě naměřené hodnoty $\mathbf{x}_{i,j}$, ale použijeme navíc funkce těchto hodnot, například s proměnnou x_1 do regrese zařadíme i x_1^2 . Zde ještě uveďme předpoklad nezávislosti stavových proměnných – v opačném případě by byla identifikace správného řešení špatně podmíněná.

Jak může být z úvodu do dynamických systémů patrné, nejčastěji se budeme snažit predikovat derivace jednotlivých stavových proměnných. Tím pádem řešíme problém

$$\underbrace{\begin{bmatrix} \dot{x}_{1,1} & \dots & \dot{x}_{1,l} \\ \vdots & \ddots & \vdots \\ \dot{x}_{n,1} & \dots & \dot{x}_{n,l} \end{bmatrix}}_{\dot{\mathbf{X}}} = \underbrace{\begin{bmatrix} x_{1,1} & \dots & x_{1,k} & f_1(\mathbf{x}_{1,\cdot}) & \dots & f_r(\mathbf{x}_{1,\cdot}) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & \dots & x_{n,k} & f_1(\mathbf{x}_{n,\cdot}) & \dots & f_r(\mathbf{x}_{n,\cdot}) \end{bmatrix}}_{\Theta(\mathbf{X})} \underbrace{\begin{bmatrix} \xi_{1,1} & \dots & \xi_{1,l} \\ \vdots & \ddots & \vdots \\ \xi_{p,1} & \dots & \xi_{p,l} \end{bmatrix}}_{\Xi} + \underbrace{\begin{bmatrix} \epsilon_1^T \\ \vdots \\ \epsilon_n^T \end{bmatrix}}_{\epsilon},$$

kde $p = k + r$, $r \in \mathbb{N}_0$, a vektorově zapsáno

$$\dot{\mathbf{X}} = \Theta(\mathbf{X})\Xi + \epsilon. \quad (1.4)$$

1.3.1 Metoda nejmenších čtverců

Nechť máme lineární model (1.3) – tedy známe kovariáty a výsledné hodnoty a chceme najít takové regresní koeficienty, že **residua** $\mathbf{e}$ budou co nejmenší, tj.

$$\min_{\xi} \|\underbrace{\mathbf{Y} - \mathbf{X}\xi}_{\mathbf{e}}\|_2^2. \quad (1.5)$$

Tato úloha lze řešit například ortogonální projekcí $\mathbf{Y}$ na $\text{im } \mathbf{X}$ nebo **metodou nejmenších čtverců** (OLS – *Ordinary Least Squares*), která spočívá v nalezení

$$\hat{\xi} = \underset{\xi}{\operatorname{argmin}} \|\mathbf{Y} - \mathbf{X}\xi\|_2^2 = \underset{\xi}{\operatorname{argmin}} (\mathbf{Y} - \mathbf{X}\xi)^T (\mathbf{Y} - \mathbf{X}\xi), \quad (1.6)$$

kde $\hat{\xi}$ je odhad regresních koeficientů. Přípustná množina je jistě otevřená (minimum hledáme na celém $\mathbb{R}^n$), a tedy pro libovolný stacionární bod $\tilde{\xi}$ účelové funkce musí platit $\operatorname{grad}_{\tilde{\xi}} \left((\mathbf{Y} - \mathbf{X}\tilde{\xi})^T (\mathbf{Y} - \mathbf{X}\tilde{\xi}) \right) = \mathbf{0}$. Z tohoto plyne, že se hledané minimum bude realizovat v některém stacionárním bodě minimalizované funkce. Proto počítejme

$$\begin{aligned} \operatorname{grad}_{\tilde{\xi}} \left((\mathbf{Y} - \mathbf{X}\tilde{\xi})^T (\mathbf{Y} - \mathbf{X}\tilde{\xi}) \right) &= \mathbf{0} \\ \operatorname{grad}_{\tilde{\xi}} (\mathbf{Y}^T \mathbf{Y} - \mathbf{Y}^T \mathbf{X}\tilde{\xi} - \tilde{\xi}^T \mathbf{X}^T \mathbf{Y} + \tilde{\xi}^T \mathbf{X}^T \mathbf{X}\tilde{\xi}) &= \mathbf{0} \\ 0 - (\mathbf{Y}^T \mathbf{X})^T - \mathbf{X}^T \mathbf{Y} + (\mathbf{X}^T \mathbf{X} + (\mathbf{X}^T \mathbf{X})^T) \tilde{\xi} &= \mathbf{0} \\ -\mathbf{X}^T \mathbf{Y} - \mathbf{X}^T \mathbf{Y} + (\mathbf{X}^T \mathbf{X} + \mathbf{X}^T \mathbf{X}) \tilde{\xi} &= \mathbf{0} \\ -2\mathbf{X}^T \mathbf{Y} + 2\mathbf{X}^T \mathbf{X}\tilde{\xi} &= \mathbf{0} \\ \tilde{\xi} &= (\mathbf{X}^T \mathbf{X})^{-1} (\mathbf{X}^T \mathbf{Y}), \end{aligned} \quad (1.7)$$

přičemž pomocí pozitivní semidefinitnosti Hessovy matice můžeme ověřit, že se jedná o minimum:

$$\nabla_{\xi}^2 \left((\mathbf{Y} - \mathbf{X}\xi)^T (\mathbf{Y} - \mathbf{X}\xi) \right) = 2\mathbf{X}^T\mathbf{X},$$

tím pádem má Hessova matice stejnou definitnost jako matice $\mathbf{X}^T\mathbf{X}$. Libovolná čtvercová symetrická matice M je pozitivně semidefinitní právě tehdy, když platí

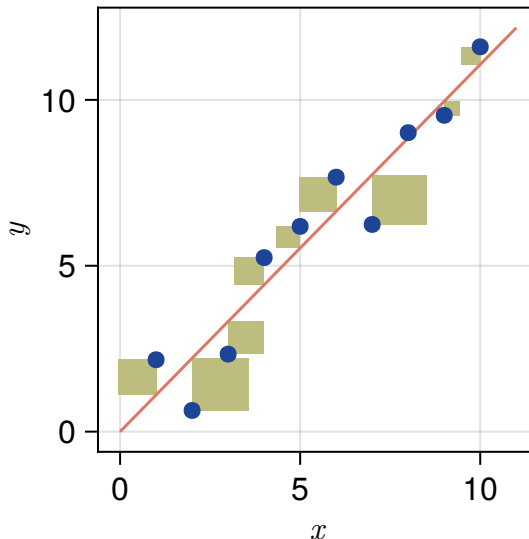
$$(\forall \mathbf{a} \in \mathbb{R}^n \setminus \{\mathbf{0}\}) \quad \mathbf{a}^T M \mathbf{a} \geq 0,$$

a tedy pro vyšetřovanou matici $\mathbf{X}^T\mathbf{X}$ a libovolný nenulový vektor $\mathbf{a} \in \mathbb{R}^n$

$$\underbrace{\mathbf{a}^T \mathbf{X}^T}_{\mathbf{b}^T} \underbrace{\mathbf{X} \mathbf{a}}_{\mathbf{b}} = \mathbf{b}^T \mathbf{b} = \|\mathbf{b}\|_2^2 \geq 0,$$

navíc si můžeme všimnout, že matice $\mathbf{X}^T\mathbf{X}$ je dokonce *pozitivně definitní*, je-li zároveň plné hodnosti. Ke stejnému závěru můžeme dospět i pomocí faktu, že $\mathbf{X}^T\mathbf{X}$ je matice kovariance.

Celkem jsme z (1.7) zjistili, že odhad ξ pomocí $\hat{\xi} = (\mathbf{X}^T\mathbf{X})^{-1} (\mathbf{X}^T\mathbf{Y})$ minimalizuje rezidua, tj. splňuje úlohu (1.5), resp. (1.6). Graficky můžeme tento proces znázornit pomocí obrázku 1.4, kde jsou vidět čtverce se stranou délky $\|\mathbf{Y} - \mathbf{X}\hat{\xi}\|_2$, které se metoda snaží minimalizovat.



Obrázek 1.4: Grafické znázornění metody nejmenších čtverců ([zdrojový kód](#))

Vraťme se nyní k úloze (1.4), což je případ l -rozměrné lineární regrese. Tu můžeme řešit tak, že postupně uděláme výše vysvětlenou lineární regresi pro každý sloupec závislé proměnné, resp. regresních koeficientů. Tuto úlohu bychom navíc mohli schematicky znázornit následujícím způsobem (uvažujeme zde 3 proměnné a s nimi i jejich druhé mocniny):

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & x_1^2 & x_2^2 & x_3^2 \end{bmatrix} \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{bmatrix} + \epsilon. \quad (1.8)$$

Zde si všimněme, že typicky bude (*skoro*) každá složka využita, tj. (*skoro*) každý regresní koeficient bude nenulový. To vychází z povahy metody nejmenších čtverců, která se snaží co nejlépe odhadnout lineární model transformující kovariáty na závislou proměnnou. To avšak zapříčiňuje, že jsou jednotlivé koeficienty často těžko interpretovatelné a dostáváme model, který nám nedává vhled do problému jako takového.

1.3.2 Řídká regrese

V předchozím případě, tj. při použití normální metody nejmenších čtverců, jsme narazili na problém s interpretovatelností, když jsou všechny možné složky využity. Nicméně pokud bychom byly schopni donutit metodu nejmenších čtverců, aby vybrala pouze ty nejdůležitější koeficienty a zbytek nechala nulový, tak tento problém z velké míry vyřešíme.

Vraťme se nyní ještě k problému (1.4), který budeme více řešit v kapitole 2. Řešení ukázky (1.8) zobecněného lineárního modelu (1.4) pomocí řídké regrese můžeme schematicky znázornit obdobným způsobem:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & x_1^2 & x_2^2 & x_3^2 \end{bmatrix} \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{bmatrix} + \epsilon,$$

což znamená, že řídkou regresí odhadujeme tvar původního autonomního systému na:

$$\begin{aligned} \dot{x}_1 &= \zeta_{2,1}x_2 + \zeta_{6,1}x_3^2 \\ \dot{x}_2 &= \zeta_{4,2}x_1^2 + \zeta_{5,2}x_2^2 \\ \dot{x}_3 &= \zeta_{1,3}x_1 + \zeta_{5,3}x_2^2 \end{aligned}$$

V tomto případě má výsledný systém 6 parametrů oproti předchozím 18, což značně zjednodušuje jakoukoliv interpretovatelnost.

LASSO Regrese

Jednou z metod řídké regrese je tzv. **LASSO regrese** (z anglického „*Least Absolute Shrinkage and Selection Operator*“), která patří mezi tzv. „smršťující metody“ (*shrinkage methods* nebo také používáme termín *sparse regression*). LASSO regrese oproti metodě nejmenších čtverců přidává ke ztrátové funkci OLS (1.5) ještě l_1 -normu¹⁹ vektoru regresních koeficientů ξ , tj.

$$\min_{\xi} \left(\frac{1}{2} \|\mathbf{Y} - \mathbf{X}\xi\|_2^2 + \mu \|\xi\|_1 \right), \quad (1.9)$$

kde μ je tzv. „vyrovnávací parametr“²⁰. Tímto LASSO penalizuje „komplexitu“ vektoru regresních koeficientů a vynucuje jeho řídkost. Pokud položíme $\mu = 0$, tak přestane mít LASSO omezení jakýkoliv vliv a dostaneme stejný odhad jako pomocí OLS. Na druhou stranu pro $\mu \rightarrow \infty$ bude vliv LASSO omezení růst a tím tlačit koeficienty $\xi \rightarrow \mathbf{0}$ (říkáme, že se „smršťují k nule“). Správným nastavením parametru μ dostaneme vektor regresních koeficientů ξ , který má většinu složek nulových – takový vektor nazýváme *řídký*²¹ [12, 13].

Fakt, že LASSO regrese vynucuje řídkost výsledného vektoru ξ , plyne i z faktu, že jde o tzv. l_1 -relaxaci obdobného problému

$$\min_{\xi} \left(\frac{1}{2} \|\mathbf{Y} - \mathbf{X}\xi\|_2^2 + \mu \|\xi\|_0 \right).$$

Pokud bychom se snažili řešit přímo tento problém, narazíme na problém, že l_0 -norma není konvexní, a tedy nemůžeme použít standardní algoritmy. Navíc oba problémy mají velmi podobná řešení a v mnoha případech dokonce shodná [14, 15].

Na rozdíl od metody nejmenších čtverců, která má analytické řešení (*closed-form solution*) (1.7), LASSO regrese v obecné formě analytické řešení **nemá**. Proto řešíme problém (1.9) pomocí iterační metody ADMM (*Alternating Direction Method of Multipliers*), což je tzv. *proximální metoda*, která je založená na **proximálním operátoru** [16].

Definice 10. Proximální operátor funkce $f : \mathbb{R}^n \rightarrow \mathbb{R}_{+\infty}$ je funkce $\text{prox}_f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ definovaná jako

$$\text{prox}_f(\mathbf{v}) = \arg\min_{\mathbf{x}} \left(f(\mathbf{x}) + \frac{1}{2} \|\mathbf{x} - \mathbf{v}\|_2^2 \right). \quad (1.10)$$

¹⁹ l_1 -norma, také nazývaná např. „taxikářská metrika“, je definovaná jako $\|\mathbf{v}\|_1 = \sum_{i=1}^n |v_i|$.

²⁰Můžeme se také setkat z anglickým označením „*sparsity-promoting knob*“.

²¹Podobný tvar ztrátové funkce jako LASSO regrese má i *ridge regrese*, která přidává l_2 -normu ξ . Ta avšak nevynucuje řídkost vektoru ξ a tedy se jí nebudeme zabývat.

Obdobně pro škálovanou funkci λf s $\lambda > 0$ bude proximální operátor

$$\text{prox}_{\lambda f}(\mathbf{v}) = \underset{\mathbf{x}}{\operatorname{argmin}} \left(f(\mathbf{x}) + \frac{1}{2\lambda} \|\mathbf{x} - \mathbf{v}\|_2^2 \right) = \underset{\mathbf{x}}{\operatorname{argmin}} \left(\lambda f(\mathbf{x}) + \frac{1}{2} \|\mathbf{x} - \mathbf{v}\|_2^2 \right),$$

což také nazýváme proximální operátor funkce f s parametrem λ . Zavedme si ADMM algoritmus nejprve pro obecný případ, tj. mějme **konvexní** funkce $f, g : \mathbb{R}^n \rightarrow \mathbb{R}_{+\infty}$ takové, že

$$(\exists \mathbf{x} \in \mathbb{R}^n) f(\mathbf{x}) < +\infty \quad \wedge \quad (\exists \mathbf{y} \in \mathbb{R}^n) g(\mathbf{y}) < +\infty,$$

a uvažujme problém

$$\min_{\mathbf{x}} (f(\mathbf{x}) + g(\mathbf{x})). \quad (1.11)$$

Potom pro (1.11) je ADMM formulována jako

$$\begin{aligned} \mathbf{x}^{k+1} &= \text{prox}_{\lambda f}(\mathbf{z}^k - \mathbf{u}^k) \\ \mathbf{z}^{k+1} &= \text{prox}_{\lambda g}(\mathbf{x}^{k+1} + \mathbf{u}^k) \\ \mathbf{u}^{k+1} &= \mathbf{u}^k + \mathbf{x}^{k+1} - \mathbf{z}^{k+1}, \end{aligned} \quad (1.12)$$

kde $\mathbf{x}^k$ značí $\mathbf{x}$ vypočtené v k -té iteraci (obdobně pro ostatní proměnné). Proměnné $\mathbf{x}^k$ a $\mathbf{z}^k$ v limitě konvergují ke stejné hodnotě (k optimu), ale mají mírně jiné vlastnosti.

Například $\mathbf{x}^k \in \mathcal{D}(f)$, ale $\mathbf{z}^k \in \mathcal{D}(g)$, a tedy pokud jsou v g zakomponovaná omezení, pak je $\mathbf{z}^k$ splňuje vždy, ale $\mathbf{x}^k$ pouze v limitě. Speciálně, je-li $g = \|\cdot\|_1$, pak dostaneme

$$\begin{aligned} \text{prox}_{\lambda \|\cdot\|_1}(\mathbf{v}) &= \underset{\mathbf{x}}{\operatorname{argmin}} \left(\lambda \|\mathbf{x}\|_1 + \frac{1}{2} \|\mathbf{x} - \mathbf{v}\|_2^2 \right) \\ &= \underset{\mathbf{x}}{\operatorname{argmin}} \left(\lambda \sum_{i=1}^n |x_i| + \frac{1}{2} \sum_{i=1}^n (x_i - v_i)^2 \right). \end{aligned}$$

$\underbrace{\hspace{10em}}_{h(\mathbf{x})}$

Všimněme si, že funkce $h(\mathbf{x})$ je *aditivně separovatelná*, tj. $h(\mathbf{x}) = h_1(x_1) + \dots + h_n(x_n)$, a tedy stačí minimalizovat každou složku zvlášť. Pro $x_i \neq 0$ bude mít i -tá složka jejího gradientu tvar

$$\begin{aligned} (\operatorname{grad}_{\mathbf{x}} h(\mathbf{x}))_i &= \frac{\partial}{\partial x_i} h_i(x_i) = \frac{\partial}{\partial x_i} \left(\lambda |x_i| + \frac{1}{2} (x_i - v_i)^2 \right) \\ &= \lambda \frac{\partial |x_i|}{\partial x_i} + x_i - v_i = x_i - v_i + \operatorname{sign}(x_i) \lambda, \end{aligned}$$

z čehož dostáváme podmínku

$$x_i - v_i + \operatorname{sign}(x_i) \lambda = 0 \iff x_i = v_i - \operatorname{sign}(x_i) \lambda.$$

Předpokládejme nyní, že $|v_i| \leq \lambda$ (pamatujme, že $\lambda > 0$), pak ale

$$\begin{aligned} \text{sign}(x_i) &= \text{sign}(v_i - \text{sign}(x_i)\lambda) \\ &= \begin{cases} \text{sign}(v_i - \lambda) = -1 & \text{pro } \text{sign}(x_i) = 1 \\ \text{sign}(v_i) & \text{pro } \text{sign}(x_i) = 0 \\ \text{sign}(v_i + \lambda) = 1 & \text{pro } \text{sign}(x_i) = -1 \end{cases} \\ &\Rightarrow \text{sign}(x_i) = \text{sign}(v_i) = 0 \\ &\Rightarrow x_i = v_i = 0, \end{aligned}$$

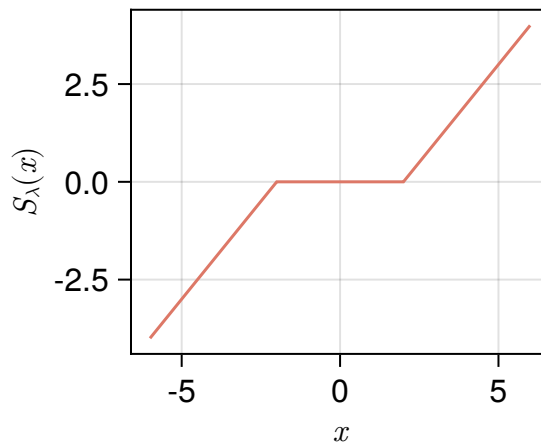
tedy $x_i \neq 0$ může být řešení pouze pokud $|v_i| > \lambda$. Z toho také plyne

$$\begin{aligned} \text{sign}(x_i) &= \text{sign}(v_i - \text{sign}(x_i)\lambda) \\ &= \begin{cases} \text{sign}(v_i - \lambda) = \text{sign}(v_i) & \text{pro } \text{sign}(x_i) = 1 \\ \text{sign}(v_i + \lambda) = \text{sign}(v_i) & \text{pro } \text{sign}(x_i) = -1 \end{cases} \\ &= \text{sign}(v_i). \end{aligned}$$

Jelikož je $h_i(x_i)$ **konvexní** funkce²² na celém $\mathbb{R}$, pak se její minimum realizuje ve stacionárním bodě, pokud takový bod existuje. Tím pádem se bude minimum pro $|v_i| > \lambda$ realizovat v $x_i = v_i - \text{sign}(v_i)\lambda$, jinak v $x_i = 0$, tj.

$$\begin{aligned} \left(\text{prox}_{\lambda\|\cdot\|_1}(\mathbf{v})\right)_i &= (S_\lambda(\mathbf{v}))_i = \begin{cases} v_i - \text{sign}(v_i)\lambda, & |v_i| > \lambda \\ 0, & |v_i| \leq \lambda \end{cases} \\ &= \text{sign}(v_i) \max\{abs(v_i) - \lambda, 0\}. \end{aligned} \quad (1.13)$$

Tento proximální operátor se nazývá *měkké prahování* (anglicky **soft-thresholding operator**) s parametrem λ a značíme ho $S_\lambda(\mathbf{v})$ [14, 17, 18].



Obrázek 1.5: Ukázka jednorozměrného měkkého prahování s $\lambda = 2$ ([zdrojový kód](#))

²²Funkce $h_i(x_i)$ je součet dvou konvexních funkcí, tedy je sama konvexní.

Pro tuto volbu $g = \|\cdot\|_1$ bude vždy $\mathbf{z}^k$ řídký vektor, zatímco $\mathbf{x}^k$ bude „pouze blízko“ vektoru $\mathbf{z}^k$ (a tedy bude $\mathbf{x}^k$ pouze „skoro řídký“ vektor) [16].

Výhodou ADMM je, že k oběma funkcím z problému (1.11) přistupujeme odděleně a pouze skrze proximální operátor – tedy ADMM je nejvýhodnější používat, pokud je jednoduché vyčíslit proximální operátory funkcí f, g , ale proximální operátor $f + g$ je složité vypočítat.

ADMM lze také interpretovat jako metodu **rozšířeného lagrangiánu** (anglicky *augmented Lagrangian*) – myšlenka spočívá v tom, že původně neohrazený problém nahradíme za ohraničený problém s penalizačním členem ve formě omezení. Pro úlohu (1.11) to znamená, že ADMM ve skutečnosti řeší problém

$$\begin{aligned} \min_{\mathbf{x}, \mathbf{z}} (f(\mathbf{x}) + g(\mathbf{z})) \\ \text{za podmínky } \mathbf{x} = \mathbf{z}, \end{aligned} \quad (1.14)$$

jehož rozšířený lagrangián je

$$\mathcal{L}_\rho(\mathbf{x}, \mathbf{z}, \mathbf{u}) = f(\mathbf{x}) + g(\mathbf{z}) + \mathbf{u}^T(\mathbf{x} - \mathbf{z}) + \frac{\rho}{2} \|\mathbf{x} - \mathbf{z}\|_2^2,$$

kde $\rho > 0$ je parametr a $\mathbf{u} \in \mathbb{R}^n$ jsou Lagrangeovy multiplikátory pro omezení $\mathbf{x} = \mathbf{z}$ (jde zároveň také o duální proměnnou k tomuto omezení) [16]. Navíc jsme schopni pomocí $\mathcal{L}_\rho$ přepsat iteraci ADMM na

$$\begin{aligned} \mathbf{x}^{k+1} &= \operatorname{argmin}_{\mathbf{x}} \mathcal{L}_\rho(\mathbf{x}, \mathbf{z}^k, \mathbf{u}^k), \\ \mathbf{z}^{k+1} &= \operatorname{argmin}_{\mathbf{z}} \mathcal{L}_\rho(\mathbf{x}^{k+1}, \mathbf{z}, \mathbf{u}^k), \\ \mathbf{u}^{k+1} &= \mathbf{u}^k + \rho(\mathbf{x}^{k+1} - \mathbf{z}^{k+1}). \end{aligned} \quad (1.15)$$

Nyní aplikujme ADMM na problém LASSO regrese (1.9) [19], přičemž jak již bylo ukázáno v (1.14), tak budeme řešit problém

$$\begin{aligned} \min_{\boldsymbol{\xi}, \mathbf{z}} \left(\frac{1}{2} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\xi}\|_2^2 + \mu \|\mathbf{z}\|_1 \right) \\ \text{za podmínky } \boldsymbol{\xi} - \mathbf{z} = 0, \end{aligned} \quad (1.16)$$

s rozšířeným lagrangiánem s parametrem ρ tvaru²³

$$\mathcal{L}_\rho(\boldsymbol{\xi}, \mathbf{z}, \mathbf{u}) = \frac{1}{2} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\xi}\|_2^2 + \mu \|\mathbf{z}\|_1 + \rho \mathbf{u}^T(\boldsymbol{\xi} - \mathbf{z}) + \frac{\rho}{2} \|\boldsymbol{\xi} - \mathbf{z}\|_2^2.$$

²³Zde jsme vynásobily Lagrangeovy multiplikátory $\mathbf{u}$ parametrem ρ , což nám zjednoduší výpočty, ačkoliv přijedeme o jistou část interpretovatelnosti $\mathbf{u}$.

Potom iterace ADMM algoritmu pro LASSO regresi je dána

$$\begin{aligned}
 \xi^{k+1} &= \underset{\xi}{\operatorname{argmin}} \mathcal{L}_\rho(\xi, \mathbf{z}^k, \mathbf{u}^k) \\
 &= \underset{\xi}{\operatorname{argmin}} \left(\frac{1}{2} \|\mathbf{Y} - \mathbf{X}\xi\|_2^2 + \mu \|\mathbf{z}^k\|_1 + \rho \mathbf{u}^{kT} (\xi - \mathbf{z}^k) + \frac{\rho}{2} \|\xi - \mathbf{z}^k\|_2^2 \right) \\
 &= \underset{\xi}{\operatorname{argmin}} \left(\begin{aligned} &\frac{1}{2} \left(\xi^T \mathbf{X}^T \mathbf{X} \xi - \overbrace{\xi^T \mathbf{X}^T \mathbf{Y}}^{\in \mathbb{R}} - \mathbf{Y}^T \mathbf{X} \xi + \mathbf{Y}^T \mathbf{Y} \right) \\ &+ \rho \mathbf{u}^{kT} (\xi - \mathbf{z}^k) + \mu \|\mathbf{z}^k\|_1 \\ &+ \frac{\rho}{2} (\xi^T \xi - \xi^T \mathbf{z}^k - \mathbf{z}^{kT} \xi + \mathbf{z}^{kT} \mathbf{z}^k) \end{aligned} \right),
 \end{aligned}$$

uvědomme si, že členy bez ξ neovlivní hodnotu minimalizovaného výrazu

$$\begin{aligned}
 &= \underset{\xi}{\operatorname{argmin}} \left(\frac{1}{2} \xi^T \mathbf{X}^T \mathbf{X} \xi - \xi^T \mathbf{X}^T \mathbf{Y} + \rho \mathbf{u}^{kT} \xi + \frac{\rho}{2} (\xi^T \xi - 2\xi^T \mathbf{z}^k) \right) \\
 &= \underset{\xi}{\operatorname{argmin}} q(\xi),
 \end{aligned}$$

tedy hledáme stacionární bod ξ^{k+1} funkce $q(\xi)$, tj. $\operatorname{grad}_\xi q(\xi^{k+1}) = \mathbf{0}$, proto počítáme

$$\begin{aligned}
 \mathbf{0} &= \frac{1}{2} (\mathbf{X}^T \mathbf{X} + (\mathbf{X}^T \mathbf{X})^T) \xi - \mathbf{X}^T \mathbf{Y} + \rho \mathbf{u}^k + \rho \xi - \rho \mathbf{z}^k \\
 &= (\mathbf{X}^T \mathbf{X} + \rho \mathbb{I}) \xi - (\mathbf{X}^T \mathbf{Y} + \rho (\mathbf{z}^k - \mathbf{u}^k)) \\
 \Rightarrow \xi^{k+1} &= (\mathbf{X}^T \mathbf{X} + \rho \mathbb{I})^{-1} (\mathbf{X}^T \mathbf{Y} + \rho (\mathbf{z}^k - \mathbf{u}^k)), \tag{1.17}
 \end{aligned}$$

přičemž $q(\xi)$ je jistě konvexní funkce, neboť je to součet konvexních funkcí²⁴. Dále

$$\begin{aligned}
 \mathbf{z}^{k+1} &= \underset{\mathbf{z}}{\operatorname{argmin}} \mathcal{L}_\rho(\xi^{k+1}, \mathbf{z}, \mathbf{u}^k) \\
 &= \underset{\mathbf{z}}{\operatorname{argmin}} \left(\frac{1}{2} \|\mathbf{Y} - \mathbf{X}\xi^{k+1}\|_2^2 + \mu \|\mathbf{z}\|_1 + \rho \mathbf{u}^{kT} (\xi^{k+1} - \mathbf{z}) + \frac{\rho}{2} \|\xi^{k+1} - \mathbf{z}\|_2^2 \right) \\
 &= \underset{\mathbf{z}}{\operatorname{argmin}} \left(\frac{\rho}{2} \|\xi^{k+1} - \mathbf{z}\|_2^2 - \rho \mathbf{u}^{kT} (\xi^{k+1} - \mathbf{z}) + \mu \|\mathbf{z}\|_1 \right) \\
 &= \underset{\mathbf{z}}{\operatorname{argmin}} \left(\frac{\rho}{2} \|(\xi^{k+1} + \mathbf{u}^k) - \mathbf{z}\|_2^2 + \mu \|\mathbf{z}\|_1 \right),
 \end{aligned}$$

z čehož podle (1.13) dostáváme

$$= S_{\frac{\mu}{\rho}}(\xi^{k+1} + \mathbf{u}^k) \tag{1.18}$$

a konečně z (1.15) plyne i poslední část jedné iterace ADMM algoritmu pro LASSO regresi (1.9)

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{x}^{k+1} - \mathbf{z}^{k+1}. \tag{1.19}$$

²⁴Člen $\xi^T \mathbf{X}^T \mathbf{Y}$ je konkávní a konvexní zároveň.

Celkem můžeme postup *ADMM* metody pro LASSO regresi zapsat v **julia** podle rovnic (1.17), (1.18), (1.19) algoritmem 1, který řeší problém (1.4), tedy l -rozměrnou LASSO regresi – tu ovšem opět provádíme pro každý sloupec závislé proměnné zvlášť [3, 19].

Sequentially Thresholded Least Squares

Druhým způsobem, jak vynutit řídkost vektoru koeficientů ξ , je metoda *STLS* (*Sequentially Thresholded Least Squares*). Tato metoda funguje na principu opakování metody nejmenších čtverců, přičemž postupně odebíráme koeficienty (a jim příslušné sloupce kovariátů), které jsou menší než námi stanovený parametr τ . Matematicky zapsáno se jedná o optimalizační problém [20]

$$\min_{\xi} \frac{1}{2} \|\mathbf{Y} - \mathbf{X}\xi\|_2^2 \quad (1.20)$$

za podmínky $\forall j \in \{1, \dots, n\} : |\xi_j| > \tau$.

Tuto metodu můžeme v **julia** zapsat pomocí algoritmu 2. Ukažme si nyní použití obou metod na ukázkovém problému

```
julia> using DataDrivenDynamics.SINDy.Optimizers # load our
↳ package
julia> Random.seed!(12345)
Random.TaskLocalRNG()
julia> X = rand(100, 10); # define the input data
julia> Y = 10 * X[:, 1] - 7 * X[:, 3] + 5 * X[:, 8] + rand(100);
julia> stls = STLS(0.3);
julia> stls(X, Y)
10-element Vector{Float64}:
 10.245952215711691
   0.0
 -6.69622459101352
   0.0
   0.0
   0.0
   0.0
   0.0
  5.247647706041146
   0.0
   0.0
julia> lasso = LASSO(2, 100);
julia> lasso(X, Y; max_iter = 1000)
10×1 Matrix{Float64}:
 10.00532610984748
   0.0
 -6.496672105858213
  0.0815879591881733
```

```

0.019081274926735947
0.15833690717414423
1.1102230246251565e-16
4.924004569131574
0.0
0.07419783265262703

```

```

Base.@kwdef struct LASSO <: Optimizer
    μ = 5
    ρ = 10
end

S(z, η) = max(z - η, 0) - max(-z - η, 0)

# Solving LASSO problem using ADMM algorithm
function (optimizer::LASSO)(θ, X̃; max_iter=100, kwargs...)
    λ, Q = eigen(θ' * θ)
    Q_inv = Matrix(Q') # Q is orthonormal
    ρ, μ = optimizer.ρ, optimizer.μ

    Ξ = zeros(variables(θ), variables(X̃))

    for variable in 1:variables(X̃)
        # Initialize all variables as zero vectors
        ξ, u, z = [zeros(variables(θ)) for _ in 1:3]

        for _ in 1:max_iter
            ξ = Q * ((Diagonal(1 ./ (λ .+ ρ)) * (Q_inv * (θ' *
                X̃[:, variable] + ρ * (z - u))))))
            z = S.(ξ + u, μ / ρ)
            u = u + ξ - z
        end

        Ξ[:, variable] = ξ
    end

    return Ξ
end

```

Algoritmus 1: Ukázková implementace LASSO regrese pomocí ADMM algoritmu

```

Base.@kwdef struct STLS <: Optimizer
    τ = 0.025
end

function (optimizer::STLS)(θ, X̃; max_iter=10, kwargs...)
    # Get the number variables in the "outcome data"
    vars = variables(X̃; kwargs...)
    # Compute OLS as the initial guess of our system
    Ξ = θ \ X̃

    for _ in 1:max_iter
        small_indices = @. abs(Ξ) < optimizer.τ
        Ξ[small_indices] .= 0

        # For each variable, do the thresholded OLS
        for variable in 1:vars
            big_indices_in = @. !small_indices[:, variable]
            Ξ[big_indices_in, variable] .= θ[:, big_indices_in]
        end
        Ξ \ X̃[:, variable]
    end

    return Ξ
end

```

Algoritmus 2: Ukázková implementace *STLS* metody

1.4 Odstraňování šumu

V ideálním světě bychom vždy pracovali s dostatkem čistých dat – čistá data neobsahují okolní vlivy či chyby v měření. To je však velmi nereálná situace, a proto je dobré se zabývat tím, jaký šum může naše data ovlivnit a jak ho odstranit či redukovat.

1.4.1 Klasifikace šumů

V této práci budeme uvažovat hlavně 3 druhy šumů, přičemž je můžeme první hrubě rozdělit na *aditivní* a *multiplikativní* šum [21]. Na vygenerování jednotlivých pozorování ze specifikovaných rozdělení pravděpodobnosti použijeme balíček [Distributions.jl](#) [22].

Aditivní šum

Nechť máme jednorozměrná²⁵ data $\mathbf{x} \in \mathbb{R}^n$ a náhodný vektor $\mathcal{N} \sim (\boldsymbol{\mu}, \sigma^2 \mathbb{I}_n)$, tj. $\mathcal{N}_i \stackrel{\text{iid}}{\sim} (\mu_i, \sigma^2)$. Nejčastěji volíme $\boldsymbol{\mu} = \mathbf{0}$. Říkáme, že $\mathbf{y} \in \mathbb{R}^n$ jsou data $\mathbf{x}$ ovlivněna *aditivním šumem* s rozptylem σ^2 tehdy, když platí

$$y_i = x_i + \mathcal{N}_i.$$

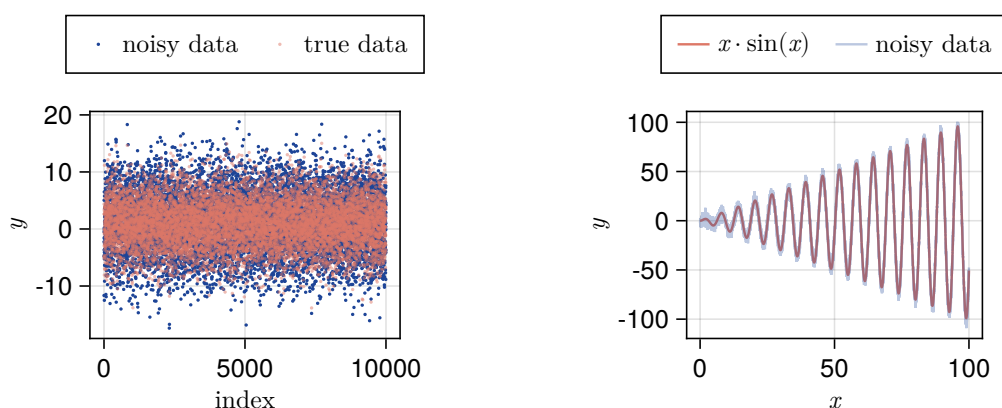
```
Base.@kwdef struct AdditiveNoise <: AbstractNoise
    distribution::Distribution = Normal(0, 0.5)

    AdditiveNoise(μ, σ) = new(Normal(μ, σ))
    AdditiveNoise(distribution::Distribution) =
        new(distribution)
end
(noise::AdditiveNoise)(data::AbstractArray) = data .+
    rand(noise.distribution, size(data))
```

Algoritmus 3: Ukázková implementace aditivního šumu

Řídí-li se šum $\mathcal{N}$ normálním rozdělením, pak ho nazýváme *Gaussovským*. Pokud je navíc rozdělení šumu konstantní pro celý vektor dat, pak jej můžeme nazvat bílým šumem – tedy v našem případě se jedná o aditivní bílý Gaussovský šum (anglicky *Additive White Gaussian Noise*). Na obrázku 1.6 můžeme vidět efekt šumu na náhodně rozmístěné body a na funkci $x \cdot \sin x$.

²⁵Uvažujeme jednorozměrná data s n pozorováními.



(a) Aditivní šum na náhodně vygenerovaných bodech

(b) Aditivní šum na funkci $x \cdot \sin x$

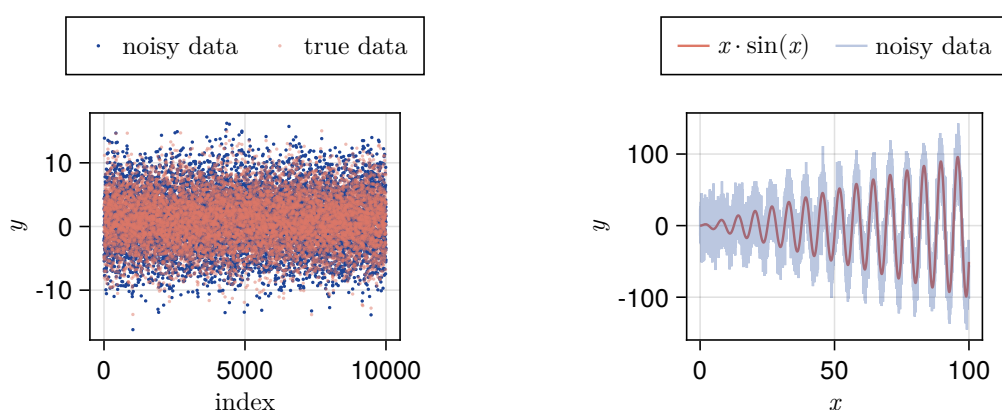
Obrázek 1.6: Ukázka vlivu aditivního šumu $\mathcal{N} \sim (\mathbf{0}, 3^2 \cdot \mathbb{I})$ ([zdrojový kód](#))

Aditivní šum s rozptylem dat

Tento šum je velmi podobný obvyklému aditivnímu šumu, jen s tím rozdílem, že místo předem daného rozptylu pro něj určujeme rozptyl podle dat, která má ovlivnit. Nechť máme „čistá“ data $\mathbf{x} \in \mathbb{R}^n$, která mají výběrový rozptyl s^2 . Potom šumem s rozptylem dat nazveme náhodný vektor s i -tou složkou $\mathcal{N}_i \sim (\mu_i, \hat{\sigma} \cdot s^2)$, kde $\hat{\sigma}$ je předem daná konstanta určující, jak velkou část rozptylu z dat šum přejme. Opět se jedná o aditivní šum

$$y_i = x_i + \mathcal{N}_i.$$

Ukázková implementace v [Julia](#) je dána algoritmem 4 a jeho efekt je zřejmý z obrázku 1.7.



(a) Aditivní šum s rozptylem dat na náhodně vygenerovaných bodech

(b) Aditivní šum s rozptylem dat na funkci $x \cdot \sin x$

Obrázek 1.7: Ukázka vlivu aditivního šumu s rozptylem dat pro $\hat{\sigma} = 0.2, \mu = \mathbf{0}$ ([zdrojový kód](#))

```

Base.@kwdef struct AdditiveVarianceNoise <: AbstractNoise
    μ::Float64 = 0
    σ̂::Float64 = 0.2
    args::Vector = []
    distribution::Type{<:Distribution} = Normal

    AdditiveVarianceNoise(μ, σ̂) = new(μ, σ̂, [], Normal)
    AdditiveVarianceNoise(μ, σ̂, args, distribution) = new(μ, σ̂,
        ↪ args, distribution)
end
function (noise::AdditiveVarianceNoise)(data::AbstractArray)
    additive_noise = AdditiveNoise(noise.distribution(
        noise.μ, sqrt(noise.σ̂) * std(data), noise.args...
    ))
    ↪ return additive_noise(data)
end

```

Algoritmus 4: Ukázková implementace aditivního šumu s rozptylem dat.

Multiplikativní šum

V posledním případě mějme jednorozměrná data $\mathbf{x} \in \mathbb{R}^n$ a náhodný vektor $\mathcal{N} \sim (\boldsymbol{\mu}, \sigma^2 \mathbb{I}_n)$, tj. $\mathcal{N}_i \stackrel{\text{iid}}{\sim} (\mu_i, \sigma^2)$, přičemž nejčastěji volíme $\boldsymbol{\mu} = \mathbf{1}$. Data $\mathbf{y} \in \mathbb{R}^n$ ovlivněná *multiplikativním* šumem s rozptylem σ^2 mají tvar

$$y_i = x_i \cdot \mathcal{N}_i.$$

Tento šum výrazně znásobuje extrémní hodnoty. Multiplikativní šum je ukázkově implementován v jazyce **julia** v algoritmu 5 a jeho efekt je patrný z obrázku 1.8.

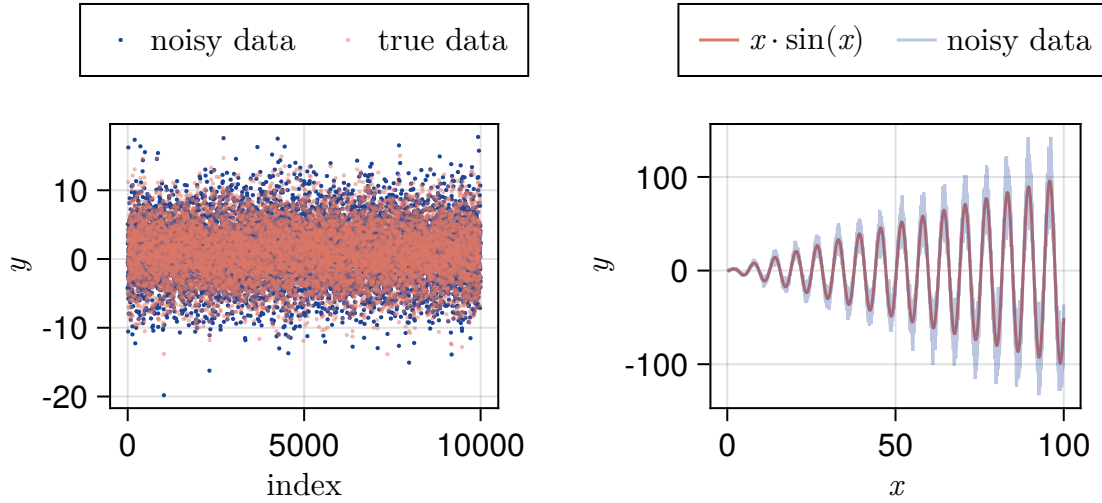
```

Base.@kwdef struct MultiplicativeNoise <: AbstractNoise
    distribution::Distribution = Normal(1, 0.2)

    MultiplicativeNoise(μ, σ) = new(Normal(μ, σ))
    MultiplicativeNoise(distribution::Distribution) =
        ↪ new(distribution)
end
(noise::MultiplicativeNoise)(data::AbstractArray) = data .*
    ↪ rand(noise.distribution, size(data))

```

Algoritmus 5: Ukázková implementace multiplikativního šumu.



(a) Multiplikativní šum na náhodně vygenerovaných bodech

(b) Multiplikativní šum na funkci $x \cdot \sin x$

Obrázek 1.8: Ukázka vlivu multiplikativního šumu pro $\mathcal{N} \sim (1, 0.2^2 \cdot \mathbb{I})$ ([zdrojový kód](#))

1.4.2 Metoda totální variace

V předchozí části jsme ukázali, s jakými druhy šumu se můžeme potkat a jak je můžeme přidat do čistých dat. Avšak většinou bude naším cílem pravý opak, tj. dostaneme data s šumem a budeme je chtít šumu zbavit. Pro tyto účely použijeme tzv. **metodu totální variace** (anglicky *total variation*).

Nechť $\mathbf{y} \in \mathbb{R}^n$ jsou jednorozměrná data obsahující šum. Definujme *totální variaci* [23]

$$V(\mathbf{y}) = \sum_{i=1}^{n-1} |y_{i+1} - y_i|, \quad (1.21)$$

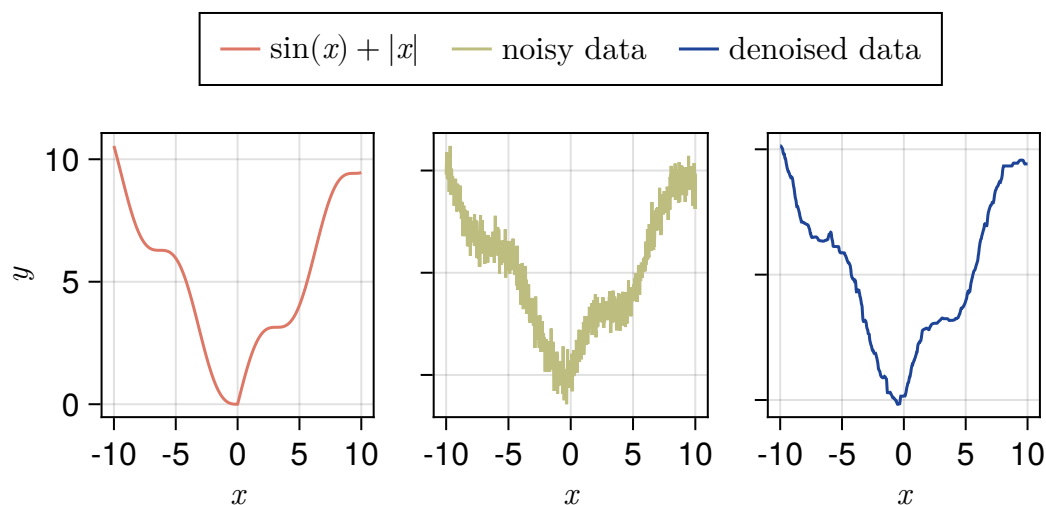
přičemž bychom chtěli najít posloupnost výstupních dat $\mathbf{x} \in \mathbb{R}^n$ takových, že mají menší totální variaci, ale přitom jsou blízko původních dat. Matematicky řečeno řešíme problém [24]

$$\min_{\mathbf{x} \in \mathbb{R}^n} \left(\eta V(\mathbf{x}) + \frac{1}{2} \|\mathbf{x} - \mathbf{y}\|_2^2 \right), \quad (1.22)$$

kde η je parametr *regularizace*, jehož „správná“ volba je sama o sobě obtížná [24, 25]. Jelikož je účelová funkce (1.22) **ostře konvexní**, tak řešení $\mathbf{x}$ tohoto problému *existuje* a je *jednoznačně určeno* vstupními daty $\mathbf{y}$.

Problém totální variace můžeme řešit například pomocí *metody pevného bodu*, *operator splitting method* [24] nebo také pomocí výše probírané metody *ADMM* [26].

V této práci dále použijeme balíček [TotalVariation.jl](#) [27], který implementuje metodu totální variace řešenou pomocí *MM metody* (*metoda minimalizace*–



Obrázek 1.9: Ukázka odstranění šumu pomocí metody totální variace ([zdrojový kód](#))

maximalizace) [28]. Ukázku odstraňování šumu se zašuměnou funkcí $\sin(x) + |x|$ můžeme vidět na obrázku 1.9.

1.4.3 Numerické derivování regularizované totální variací

V předchozí sekci jsme se bavili o odstraňování šumu z dat pomocí metody totální variace. Pokud si vzpomeneme na formulaci problému (1.4), tak si můžeme uvědomit, že na vyřešení, tj. nalezení parametrů systému, potřebujeme znát i derivace v každém časovém okamžiku.

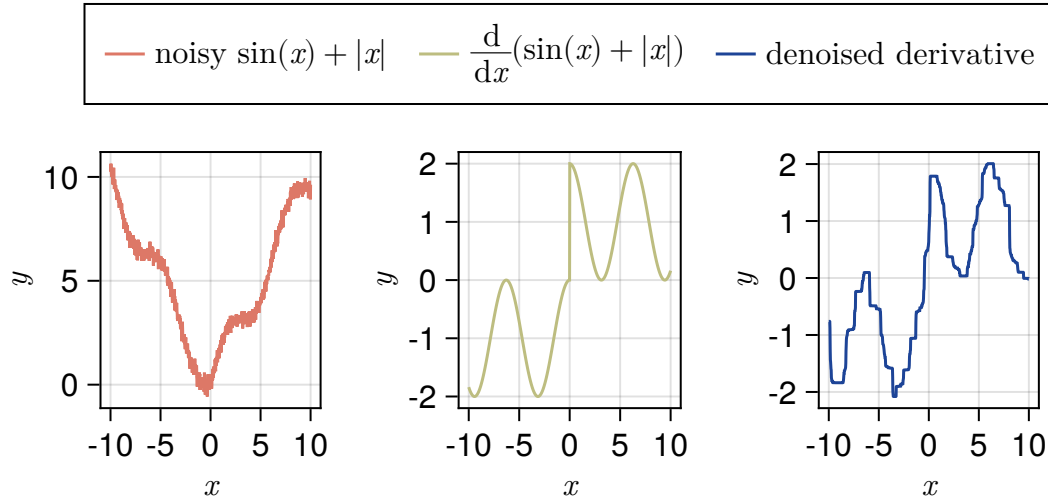
V reálném světě se často můžeme dostat do situace, kdy známe jednotlivé stavové proměnné, ale nikoliv jejich derivace – tedy je musíme získat pomocí numerických metod. Avšak častým problémem numerického derivování, zejména metod *konečných diferencí*, bývá chování na datech ovlivněných šumem, přičemž odstranění šumu před aplikováním těchto metod nezřídka nedává uspokojivé výsledky [29].

Z tohoto důvodu v této práci použijeme *metodu numerického derivování regularizované totální variací* (anglicky *Total Variation Regularized Numerical Differentiation (TVDiff)*), implementovanou v balíčku [NoiseRobustDifferentiation.jl](#) [29].

Tento problém můžeme řešit pomocí *Tichonovovy regularizace* odpovídajícího inverzního problému [30]. Hledejme prozatím derivaci funkce $f : \mathbb{R} \rightarrow \mathbb{R}$ na intervalu $[0, L]$. Tato derivace je jistě řešením funkcionálního problému

$$\min_u (\eta R(u) + DF(A(u) - f)), \quad (1.23)$$

kde u je funkce $u : \mathbb{R} \rightarrow \mathbb{R}$, $R(u)$ je regularizační (*penalizační*) člen, $A(u)$ je operátor *antidiference*, tj. $A(u) = \int_0^L u(x) \, dx$, $DF(v)$ je člen zachycující *věrohodnost k datům* (anglicky *data-fidelity term*), který penalizuje rozdíl mezi $A(u)$ a f , a η je parametr regularizace, který vyvažuje tyto dva členy.



Obrázek 1.10: Ukázka numerického derivování regularizovaného metodou totální variace ([zdrojový kód](#))

Jako DF nejčastěji používáme l_2 -normu, o které lze ukázat, že je „dobrou volbou“, máme-li aditivní bílý Gaussovský šum²⁶. Dále jako $R(u)$ volíme *totální variaci*, tj.

$$R(u) = \int_0^L |u'(x)| \, dx,$$

tedy úlohu (1.23) můžeme v našem případě formulovat jako

$$\min_u \left(\alpha \left(\int_0^L |u'(x)| \, dx \right) + \frac{1}{2} \left(\int_0^L |A(u(x)) - f(x)|^2 \, dx \right) \right), \quad (1.24)$$

kde $u : \mathbb{R} \rightarrow \mathbb{R}$. Uvažujme nyní diskrétní případ, tj. máme vstupní data $\mathbf{f} \in \mathbb{R}^n$ taková, že $\forall i \in \{1, \dots, n\} : \mathbf{f}_i = f(x_i)$ pro vhodné $x_i \in \mathbb{R}$. Poté budeme řešit místo (1.24) problém

$$\min_{\mathbf{u} \in \mathbb{R}^n} \left(\eta V(\mathbf{u}) + \frac{1}{2} \|A_d(\mathbf{u}) - \mathbf{f}\|_2^2 \right), \quad (1.25)$$

kde $V(\mathbf{x})$ je totální variace (1.21) a funkce $A_d(\mathbf{u})$ aproximuje integrál funkce $u(x)$ na bodech $\mathbf{u}$. Neboť A_d můžeme brát jako metodu *numerického integrování*, tak se nabízí použít například Gaussovu nebo Newtonovu–Cotesovu kvadraturní formuli, přičemž nejčastěji používáme tzv. *lichoběžníkové pravidlo* [31, 29].

Tento problém lze řešit například metodou největšího spádu nebo lépe tzv. algoritmem s opožděnou difuzivitou (anglicky *lagged diffusivity algorithm*). Ukázku regularizovaného numerického derivování se zašuměnou funkcí $\sin(x) + |x|$ můžeme vidět na obrázku 1.10.

²⁶Jiné typy šumů by vyžadovaly jinou volbu této funkce.

Kapitola 2

Řídká identifikace nelineární dynamiky (SINDy)

Jak již bylo nastíněno v kapitole 1, hlavním tématem této práce jsou metody nalézání rovnic dynamických systémů z dat. V této kapitole se podíváme na řídkou identifikaci nelineární dynamiky (SINDy) (z anglického *Sparse Identification of Nonlinear Dynamics*), se kterou přišel prof. Brunton et al. v roce 2016 [32, 1]. Tato kapitola je z velké části založena právě na jeho publikacích [1, 33].

Ačkoliv zde budeme používat výhradně SINDy, patří se uvést alternativní metody [34, 1]. Mezi ně patří například *symbolická regrese* [35], *automatická inference dynamiky* [34] nebo nově i pomocí *aritmetických neuronových sítí* [36].

I když se zde budeme zabývat pouze identifikací *obyčejných diferenciálních rovnic*, SINDy algoritmus lze formulovat i pro *parciální diferenciální rovnice* [1, 37].

2.1 Teoretický úvod

Předpokládejme, že zkoumaný dynamický systém

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \quad (2.1)$$

má pouze malé množství „aktivních členů“ v prostoru všech možných členů pravé strany této rovnice. Například FitzHughův–Nagumův model (1.2) obsahuje pouze lineární členy a třetí mocniny v každé rovnici.

Jinak řečeno se snažíme aproximovat neznámou funkci $\mathbf{f}(\mathbf{x})$ *zobecněným lineárním modelem*

$$\mathbf{f}(\mathbf{x}) \approx \sum_{i=1}^p \theta_i(\mathbf{x}) \zeta_i = \mathbf{\Theta}(\mathbf{x}) \boldsymbol{\zeta}, \quad (2.2)$$

přičemž se snažíme minimalizovat počet nenulových členů v $\boldsymbol{\zeta}$, kde $p \in \mathbb{N}$, $\theta_i : \mathbb{R}^n \rightarrow \mathbb{R}$ a také

$$\mathbf{\Theta}(\mathbf{x}) = \begin{pmatrix} \theta_1(\mathbf{x}) \\ \vdots \\ \theta_p(\mathbf{x}) \end{pmatrix}^T, \quad \boldsymbol{\zeta} = \begin{pmatrix} \zeta_1 \\ \vdots \\ \zeta_p \end{pmatrix}. \quad (2.3)$$

Funkce $\theta_k(\mathbf{x})$ pro $k = 1, \dots, p$ dohromady nazýváme **knihovnou kandidátů** (či *bázi kandidátních prvků*). Pomocí řídké regrese 1.3.2 jsme schopni najít požadované ξ pro (2.2) takové, že bude z většiny tvořeno nulovými členy, tj. bude *řídké*.

Nechť máme data z časové řady příslušné systému (2.1), které uspořádáme do matice

$$\mathbf{X} = [\mathbf{x}(t_1) \quad \mathbf{x}(t_2) \quad \dots \quad \mathbf{x}(t_n)]^T$$

a obdobně vytvoříme matici derivací

$$\dot{\mathbf{X}} = [\dot{\mathbf{x}}(t_1) \quad \dot{\mathbf{x}}(t_2) \quad \dots \quad \dot{\mathbf{x}}(t_n)]^T.$$

V praxi se často setkáváme se situací, kdy máme matici $\mathbf{X}$ ovlivněnou šumem, a neřídka matici $\dot{\mathbf{X}}$ vůbec neznáme a musíme ji odvodit z dat. Tyto problémy řešíme pomocí metod *totální variace* představených v sekci 1.4.2, respektive v sekci 1.4.3 [1].

Knihovnu kandidátních funkcí konstruujeme z dat uložených v matici $\mathbf{X}$ jako

$$\Theta(\mathbf{X}) = [\mathbf{1} \quad \mathbf{X} \quad P_2(\mathbf{X}) \quad \dots \quad P_d(\mathbf{X}) \quad \dots \quad \sin(\mathbf{X}) \quad \dots \quad \mathbf{f}(\mathbf{X})], \quad (2.4)$$

kde $P_d(\mathbf{X}) = (p_d(\mathbf{X}_{1,\cdot}) \quad \dots \quad p_d(\mathbf{X}_{n,\cdot}))^T$, přičemž $p_d : \mathbb{R}^k \rightarrow \mathbb{R}$ značí polynom k -proměnných d -tého stupně, $\sin(\mathbf{X})$ je myšlen po prvcích (stejně jako obdobné funkce) a nakonec $\mathbf{f} : \mathbb{R}^{n \times k} \rightarrow \mathbb{R}^n$ libovolná námi zvolená funkce – obecně mohou být součástí knihovny kandidátů libovolné funkce, o kterých předpokládáme, že by se v systému (2.1) mohly nacházet.

Celkem tedy SINDy aproximuje systém (2.1) pomocí *zobecněného lineárního modelu* (1.4)

$$\dot{\mathbf{X}} = \Theta(\mathbf{X})\Xi + \epsilon,$$

kde i -tý sloupec ξ_i matice regresních koeficientů Ξ odpovídá koeficientům aproximace i -té složky systému (2.1) ve smyslu (2.2). *Šetrným modelem* (z anglického *parsimonious model*) rozumíme takový model, který bude dobře aproximovat původní systém a zároveň obsahovat pouze malé množství nenulových koeficientů v Ξ . Obecně můžeme tento problém formulovat jako

$$\min_{\Xi} \left(\frac{1}{2} \|\dot{\mathbf{X}} - \Theta(\mathbf{X})\Xi\|_2^2 + \mu R(\Xi) \right), \quad (2.5)$$

kde $R(\cdot)$ je regularizující funkce vynucující řídkost (viz také (1.9) speciálně pro LASSO regresi). Šetrný model můžeme získat například pomocí metody *STLS*, viz 1.3.2, nebo pomocí LASSO regrese, viz 1.3.2, přičemž prof. Brunton doporučuje metodu *STLS* [1]. Dále pak v [38] je ukázáno použití metody *SR3*, jakožto alternativy k *STLS*, viz 2.4.4, a v sekci 2.4.4 navrhuje novou metodu *DSTLS* takéž upravující původní metodu *STLS*.

Dohromady aproximaci i -té složky systému (2.1) dostaneme jako

$$\dot{x}_i = \Theta(\mathbf{x})\xi_i,$$

kde $\Theta(\mathbf{x})$ vnímáme jako vektor *symbolických funkcí* ve smyslu (2.3).

2.2 Ukázková implementace

Ukažme si nyní ukázkovou implementaci SINDy v jazyce **julia**. Jako optimalizační algoritmy využijeme *STLS*, algoritmus 2, a LASSO regresi, algoritmus 1, z kapitoly 1.3.2 – tyto metody budeme souhrnně v kódu, viz algoritmus 6, označovat abstraktním typem `Optimizer`, přičemž volba konkrétní metody bude na uživateli.

Hlavní myšlenkou SINDy je první vytvořit knihovnu možných funkcí, pro kterou budeme používat abstraktní typ `Library`, a potom se pokusit najít takové koeficienty pro funkce z knihovny, že co nejlépe vystihují data ze systému (2.1) a přitom jsou řídké.

Tento postup můžeme algoritmicky zapsat pomocí funkce `discover`, viz algoritmus 6, která na vstupu potřebuje matici X , maticí derivací $\dot{X}$, knihovnu kandidátů a optimalizační metodu.

```
function discover(X,  $\dot{X}$ , library::T, optimizer::TT; kwargs...)
    where {T<:Library, TT<:Optimizer}
     $\theta$  = library(X; kwargs...)
     $\Xi$  = optimizer( $\theta$ ,  $\dot{X}$ ; kwargs...)
    return  $\Xi$ 
end
```

Algoritmus 6: Ústřední funkce SINDy metody

Knihovna funkcí, která je v algoritmu 6 vyjádřena abstraktním typem `Library`, může mít mnoho různých podob a tím se bude lišit, jak ji budeme v kódu vytvářet.

Pro začátek si definujme funkci, která pro každý řádek matice X vyhodnotí knihovní funkce na tomto pozorování.

```
function (library::Library)(X; iterate_by=:eachrow, kwargs...)
     $\theta$  = reshape([func(X) for func in library],
    length(iterate_by(X)), 1)
    return hcat( $\theta$ ...)
end

eachstate(func, X; iterate_by=:eachrow) = [func(iteration) for
    iteration in iterate_by(X)]
```

Algoritmus 7: Ústřední funkce pro práci s knihovnami kandidátů

Vlastnoručně implementujeme pouze několik vybraných typů knihoven, a to

monomiální knihovnu (viz Přílohy/algoritmus 11), např. funkce $(1, x, y, x^2, y^2)$, polynomiální knihovnu (viz Přílohy/algoritmus 12), např. funkce $(1, y, yx, x)$, a knihovnu z libovolných funkcí zvolených uživatelem (viz Přílohy/algoritmus 10). Dalšími knihovnami, které bychom mohli použít, by byly například trigonometrické funkce.

Tímto jsme poskládali dohromady všechny stavební bloky metody SINDy a můžeme si ji ukázat na příkladu systému daném FitzHughovým–Nagumovým modelem

$$\begin{aligned}\dot{V} &= 0.8 - W + V - \frac{1}{3}V^3, \\ \dot{W} &= 0.056 - 0.064 \cdot W + 0.08 \cdot V.\end{aligned}\tag{2.6}$$

Pro jednoduchost předpokládejme, že už máme data o tomto systému, přesněji matici měření stavových proměnných $\mathbf{X}$ a matici derivací $\dot{\mathbf{X}}$. Dále určíme, že budeme používat knihovnu polynomů 2 proměnných stupně nejvýše 3 a jakožto optimalizační metodu využijeme *STLS* ([zdrojový kód](#)).

```
julia> basis = PolynomialLibrary(3, 2)
DataDrivenDynamics.SINDy.Libraries.PolynomialLibrary{Int64}(3, 2)
julia> optimizer = STLS()
DataDrivenDynamics.SINDy.Optimizers.STLS(0.025)
julia> Ξ = discover(X, Xdot, basis, optimizer; max_iter=100)
10x2 Matrix{Float64}:
 0.8      0.056
-1.0     -0.064
 0.0       0.0
 0.0       0.0
 1.0      0.08
 0.0       0.0
 0.0       0.0
 0.0       0.0
 0.0       0.0
-0.333333  0.0
julia> prettyprint(Ξ, basis; vars=["V" "W"])
Found equations for the system are:
  Ḃ = 0.80 + -1.00*W + 1.00*V + -0.33*V^3.0
  Ḃ = 0.06 + -0.06*W + 0.08*V
```

Z ukázky výše vidíme, že až na zaokrouhlovací chyby jsme v tomto, ačkoliv jednoduchém, případě našli systém správně.

2.3 Knihovna DataDrivenDiffEq.jl

Ačkoliv je pro vzdělávací účely vlastní implementace nutností, z pohledu dlouhodobé udržitelnosti, efektivity či funkcionality je vhodné použít volně dostupné

knihovny. Knihovna `DataDrivenDiffEq.jl` [20], přesněji „podbalíček“ `DataDrivenSparse`, implementuje nejen metodu SINDy v jazyce `julia`. Udělejme proto nyní rychlé srovnání naší vlastní implementace a této knihovny.

Abychom mohli použít SINDy metodu v této knihovně, je nejdříve potřeba si zadefinovat tzv. `DataDrivenProblem`. V této práci budeme výhradně používat `ContinuousDataDrivenProblem`, ale z dokumentace můžeme zjistit, že je implementovaný např. i `DiscreteDataDrivenProblem` a další.

```
julia> using DataDrivenDiffEq, DataDrivenSparse
julia> @named data_driven_problem =
  ~ ContinuousDataDrivenProblem(X', times, X')
Continuous DataDrivenProblem{Float64} data_driven_problem in 2
  ~ dimensions and 1001 samples
```

Dalším krokem je volba knihovny kandidátů, což zde probíhá zvolením proměnných přes makro `@variables`. V našem případě jsme dále s pomocí funkce `polynomial_basis` vytvořili pole obsahující funkce polynomiální báze v proměnných V, W stupně nejvýše 3. V posledním kroku tuto bázi ještě převedeme na odpovídající typ `Basis`, přičemž v tomto místě bychom mohli upravit poslední detaily ohledně této knihovny kandidátů.

```
julia> @variables V, W
2-element Vector{Symbolics.Num}:
 V
 W
julia> term_library = polynomial_basis([V, W], 3)
10-element Vector{Symbolics.Num}:
 1
 V
 V^2
 V^3
 W
 V*W
 W*(V^2)
 W^2
 V*(W^2)
 W^3
julia> @named model_basis = Basis(term_library, [V, W])
Model model_basis with 10 equations
States (2):
 V
 W
Parameters (0):
```

V neposlední řadě potřebujeme na vyřešení SINDy problému (1.4) optimalizační metodu. Knihovna `DataDrivenDiffEq.jl` poskytuje metodu `STLS` (pod typem

STLSQ), metodu LASSO regrese řešenou pomocí ADMM algoritmu (pod typem ADMM) a také metodu SR3 (pod typem SR3), kterou dále rozebereme v sekci 2.4.4.

```
julia> optimizer = STLSQ(0.025)
STLSQ(0.025)
```

V tuto chvíli nám již zbývá na vyřešení problému zavolat funkci `solve`, která odpovídá funkci `discover` z naší implementace. Pomocí funkce `get_basis` získáme nalezené rovnice, přičemž hodnoty parametrů řešení dostaneme funkcí `get_parameter_map`, která vrací jejich *hash mapu* ([zdrojový kód](#)).

```
julia> solution = solve(data_driven_problem, model_basis,
    ↪ optimizer); print(solution)
"DataDrivenSolution{Float64}" with 2 equations and 7 parameters.
Returncode: Success
Residual sum of squares: 3.003304826639461e-17
julia> fitted_system = get_basis(solution); print(fitted_system)
Model ##Basis#319 with 2 equations
States : V W
Parameters : 7
Independent variable: t
Equations
Differential(t)(V) = p1 + V*p2 + W*p4 + p3*(V^3)
Differential(t)(W) = p5 + V*p6 + W*p7
julia> fitted_parameters = get_parameter_map(fitted_system)
7-element Vector{Pair{SymbolicUtils.Sym{Real,
    ↪ Base.ImmutableDict{DataType, Any}}, Float64}}:
p1 ⇒ 0.8
p2 ⇒ 1.0
p3 ⇒ -0.3333333333
p4 ⇒ -1.0
p5 ⇒ 0.056
p6 ⇒ 0.08
p7 ⇒ -0.064
```

Celkem si můžeme všimnout, že jsem dostali stejné řešení jako pomocí naší vlastní implementace.

2.4 Výzvy při hledání modelů

V této kapitole jsme zatím předváděli ukázkový případ použití SINDy metody na FitzHughově–Nagumově modelu. Předpokládali jsme, že známe matici stavových proměnných X , matici derivací $\dot{X}$, že jsou tyto hodnoty bez šumu a že jich známe dostatek. Navíc jsme mlčky uvažovali, že měříme *ty správné* proměnné, a volbu kandidátů do knihovny jsme udělali na základě předchozí znalosti tohoto

systému – věděli jsme, že obsahuje nejvýše polynom 3. stupně, a tedy jsme jej do knihovny zahrnuli. Naopak jsme knihovnu kandidátů ochudili o funkce, o kterých jsme věděli, že se v systému nevyskytují.

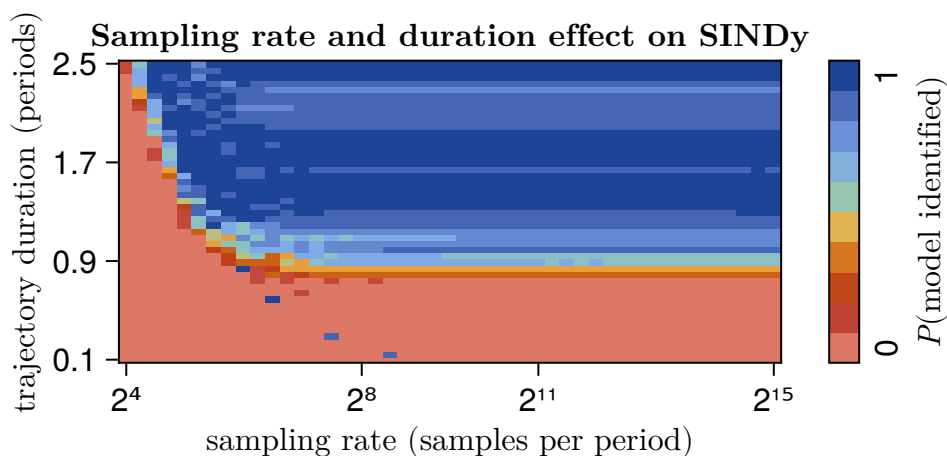
Avšak tyto předpoklady jistě nemusí být splněny (či alespoň část z nich). Proto se nyní budeme věnovat tomu, jaké mají zmíněné nedostatky dat vliv na chování SINDy metody a jak je řešit.

2.4.1 SINDy a data

Jak již bylo nastíněno na začátku této sekce, pojďme se nyní zaměřit na možné problémy spojené s nedostatkem dat. Tedy nyní předpokládejme, že měříme správně proměnné a máme čistá data.

Aby SINDy metoda fungovala dle očekávání, je potřeba, abychom měli *relativně* *jemně* měřená data po *dostatečně dlouhou* dobu [39]. Co to ale znamená v praxi?

V článku [40] dr. Kathleen Champion a prof. Brunton ukázali, že pokud máme data měřená dostatečně často (vůči jedné periodě¹ v datech), tak stačí i méně než jedna perioda na korektní identifikaci systému, podle tvaru systému a toho, zda je deterministicky chaotický². Pro Lorenzův model (viz sekce 3.2) je to cca 70 – 110 % periody, podle počáteční podmínky, viz obrázek 2.1.



Obrázek 2.1: Vliv délky trajektorie a počtu pozorování na jednu periodu na chování SINDy metody na Lorenzově modelu s chaotickými parametry ([zdrojový kód](#))

V opačném případě, tj. pokud „snímkuje“ data po dlouhých časových okamžicích, potřebujeme „vidět“ celý systém i několikrát. Pro úplnost uvedme, že za úspěšně nalezený model jsme považovali situaci, kdy jsme našli model se 7 parametry a žádný parametr se od původního nelišil o více než 0.01. Každou dvojici

¹Periodou se např. u Lorenzova systému rozumí doba, kterou trajektorie potřebuje na „projití jednoho křídla“ a odpovídá $t \approx 0.759$.

²Systém nazveme *deterministicky chaotický*, jestliže velmi malá změna v počátečních podmínkách způsobí velkou změnu v dynamice tohoto systému.

délky trajektorie a počtu pozorování na jednu periodu jsme spustili pro 10 různých počátečních podmínek (ty byly náhodně vybrány z bodů na atraktoru před spuštěním programu). Při hledání modelu jsme použili polynomiální knihovnu až 5. stupně a optimalizační metodu *STLS* s $\tau = 0.1$.

Avšak i s dostatkem dat se nevyhneme časté situaci, že jsou tato data zatížena šumem a navíc nemáme měření derivací stavových proměnných. V takovém případě musíme použít techniky odstranění šumu, abychom podpořili úspěšné nalezení dynamiky.

Na odstranění šumu z dat můžeme použít například metodu totální variace, definovanou v sekci 1.4.2. Na vypočtení derivací z dat zatížených šumem je výhodné použít některou *regularizovanou metodu numerického derivování*, např. pomocí totální variace [40], viz sekce 1.4.3.

Ukažme si nyní vliv velikosti šumu na počet nalezených parametrů, R^2 , AIC a BIC³ Lorenzova systému, viz obrázky 2.3 a 2.4. Ačkoliv jsme ještě nerozebrali zobrazené optimalizační metody (viz sekce 2.4.4), obrázky 2.3 a 2.4 pěkně ilustrují vliv šumu a proto je uvádíme již nyní. Pro každou hodnotu $\hat{\sigma}$ velikosti aditivního šumu s rozptylem dat a každou optimalizační metodu byly všechny „diagnostické metriky“ spočteny pro 10 různých – předem určených – počátečních podmínek. Tedy na obrázcích 2.3 a 2.4 tlusté čáry značí průměr z výsledků pro všechny počáteční podmínky a světleji zbarvené pásy znázorňují rozpětí minima a maxima těchto výsledků.

Další metodou, jak se vypořádat s šumem, je v jistém smyslu oslabení SINDy metody [41]. Vzpomeňme si, že SINDy spočívá v nalezení řídkého vektoru ξ tak, že knihovna kandidátů $\Theta(\mathbf{x})$ dobře aproximuje původní systém (2.1), viz (2.2)

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \approx \sum_{j=1}^p \theta_j(\mathbf{x}) \xi_j = \Theta(\mathbf{x}) \xi,$$

což pro i -tou složku znamená

$$\dot{x}_i = (\Theta(\mathbf{x}) \xi)_i = \sum_{j=1}^p \xi_j \theta_{j,i}(\mathbf{x}),$$

kde $\theta_{j,i}(\mathbf{x}) = (\theta_j(\mathbf{x}))_i$. Tuto formulaci ale můžeme relaxovat a místo ní řešit problém

$$x_i(t) = x_i(0) + \sum_{j=1}^p \xi_j d_{j,i}(\mathbf{x}, t),$$

přičemž definujeme

$$d_{j,i}(\mathbf{x}, t) = \int_0^t \theta_{j,i}(\mathbf{x}(\tau)) d\tau.$$

Tento přístup umožňuje SINDy tolerovat větší *aditivní* šum, neboť jej „rozprostře“ přes všechna data [41].

³Akaikeho a Bayesovské informační kritérium, více [zde](#) a [zde](#).

2.4.2 Souřadnicový systém

Dosud jsme předpokládali, že měřené proměnné (sloupce matice $\mathbf{X}$) odpovídají proměnným hledaného systému (2.1)

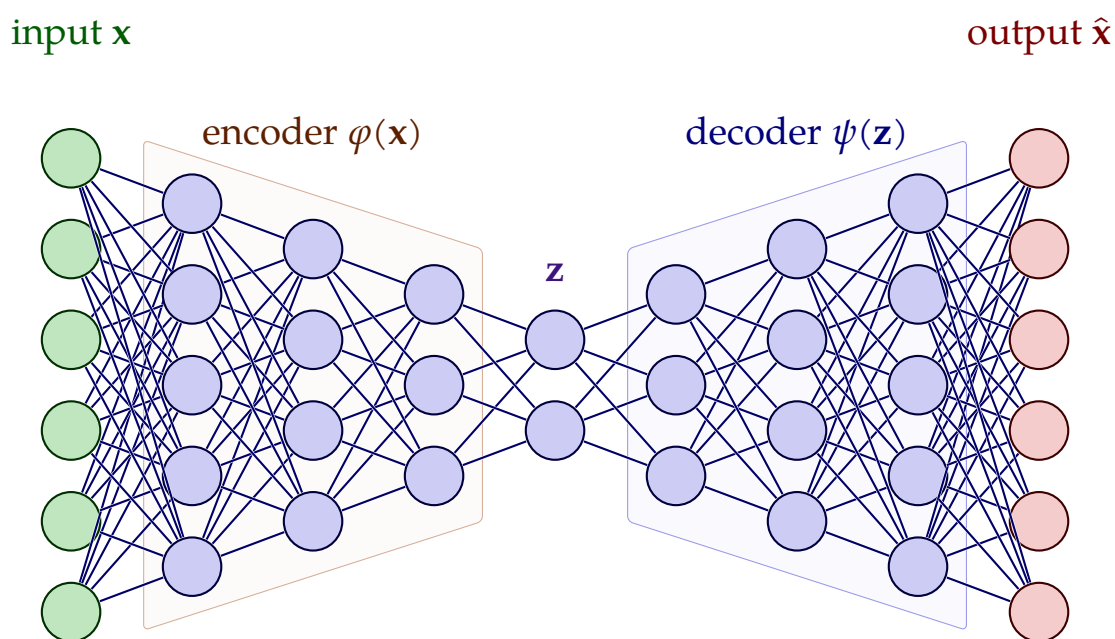
$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$$

Jak ale můžeme vědět, že měřené proměnné jsou „ty správné“? V reálném světě můžeme narazit na různé problémy [42], např.

- měříme pouze některé proměnné systému (2.1),
- měříme „příliš mnoho proměnných“ – tedy mezi některými mohou být silné korelace,
- v měřených proměnných nemá systém (2.1) řídkou reprezentaci.

Například na prostorově časových systémech, jako je třeba proudění za překážkou, na volbu souřadnicového systému dobře funguje SVD rozklad, který poskytuje ortogonální bázi dobře vystihující variabilitu dat [40, 43].

SVD navíc můžeme vnímat jako jednovrstvý lineární autoenkodér – neuronovou síť, která mnohorozměrný vstup transformuje přes „malou“ vrstvu zpátky na mnohorozměrný výstup. Tento koncept můžeme zobecnit a místo jednovrstvého lineárního autoenkodéru použít vícevrstvou nelineární neuronovou síť, viz obrázek 2.2.



Obrázek 2.2: Schematické znázornění autoenkodéru pro použití se SINDy metodou

Takto lze zkombinovat schopnost autoenkodéru naučit se vhodný souřadnicový systém se SINDy hledající řídký systém na těchto nových souřadnicích. Dokonce je optimalizaci SINDy problému možné zařadit do ztrátové funkce neuronové sítě autoenkodéru [43].

Obdobný postup můžeme použít i na situaci, kdy vstupní data o systému $\mathbf{X}$ neobsahují některou z proměnných, pokud tento systém splňuje jisté požadavky – mimo jiné předpoklady [Takensovy věty o vnoření](#). Pomocí *time-delay embedding* (volně přeloženo *vnoření se zpožděním*) souřadnic, která transformujeme autoenkodérem, jsme schopni nalézt řídký model v „originálních souřadnicích“⁴ [43].

2.4.3 Výběr členů do knihovny

Výběr členů do knihovny kandidátních funkcí z velké části ovlivňuje úspěch SINDy metody – pokud knihovna nebude obsahovat členy, které byly v původním systému (2.1), pak se nám nejspíše nepodaří najít řídký model, který by tato data dobře aproximoval [44].

Určení těchto funkcí může být obtížné, nemáme-li žádné znalosti o příslušném fyzikálním (či jiném) systému, a jednoduše se můžeme dostat do situace, že nevíme, které členy by knihovna obsahovat měla a které ne. Obecně můžeme narazit na následující výzvy:

Podmíněnost matice $\Theta(\mathbf{X})$

Knihovna by měla být navržena tak, aby zahrnovala členy, které jsou vzájemně nezávislé nebo jen málo korelované, a aby se minimalizovalo množství redundance a zbytečné komplexity v modelu. V opačném případě bude pravděpodobně matice $\Theta(\mathbf{X})$ špatně podmíněná a bude náročné najít minimalizující matici Ξ , zvláště pokud budou data ovlivněna šumem. Například pokud knihovna obsahuje funkce $\sin x, x, x^3, x^5$, pak je velká šance, že matice $\Theta(\mathbf{X})$ bude špatně podmíněná, neboť jsou tyto členy *nezanedbatelně korelované*. S tímto se také váže předpoklad, že stavové proměnné systému jsou nezávislé – v opačném případě bychom ze stejného problému museli některé vyřadit.

Je vhodné zmínit, že pod podmíněností matice myslíme *ERC podmíněnost* [45, 44], která kvantifikuje podmíněnost matice na jejích řídkých podprostorech.

Velikost knihovny

Pokud máme příliš rozsáhlou knihovnu, také se dostáváme ke problému zmíněnému výše. Nejen, že pro velké matice je výpočetně náročné spočítat pseudoinverzi, ale také je matice $\Theta(\mathbf{X})$ příslušná příliš velké knihovně špatně podmíněná a dochází k numerickým chybám při výpočtu pseudoinverze.

Nelinearita systému

Metoda SINDy je navržena pro identifikaci lineárních modelů nelineárních systémů. Pokud je však systém příliš nelineární (např. pokud je zadán racionálními lomenými funkcemi), může být těžké najít vhodnou knihovnu členů pro identifikaci modelu.

⁴Ve vnořených souřadnicích často dostaneme lineární model, což souvisí s teorií Koopmanova operátoru.

Vnější vlivy

Pokud máme systém, který závisí nejen na stavových proměnných, ale i na vnějších vlivech, pak bychom měli možné formy tohoto vlivu zakomponovat do knihovny kandidátů. Příkladem může být model predátor–kořist, ve kterém kontrolujeme populaci např. kořisti, či epidemiologické modely, o kterých máme data z reálného světa, kde si nemůžeme dovolit přestat léčit lidi a nebo zrušit všechna bezpečnostní opatření, jen abychom mohli změřit dynamiku systému bez okolních vlivů.

2.4.4 Optimalizační algoritmy

Vzpomeňme si, že podstatou SINDy metody je najít koeficienty Ξ šetrného modelu řešícího úlohu (1.4)

$$\dot{\mathbf{X}} = \Theta(\mathbf{X})\Xi + \epsilon.$$

Takový model můžeme dostat různými optimalizačními metodami, které si dále rozebereme. Jejich srovnání můžeme vidět i na obrázcích 2.3 a 2.4.

LASSO regrese

Ačkoliv by se intuitivně mohlo zdát, že LASSO regrese (viz 1.3.2) bude dávat správné výsledky (např. i v [1]), ve skutečnosti tomu tak většinou není. Matice koeficientů Ξ podle LASSO regrese často obsahuje velké množství malých, ale nenulových koeficientů. I když zvýšíme parametr regularizace μ , tak často neidentifikujeme správný model [38, 46].

Sequentially Thresholded Least Squares (STLS)

Tato optimalizační metoda, doporučená v [1], je založena na opakování metody nejmenších čtverců při vyřazování příliš malých koeficientů, viz 1.3.2. Pro „standardní“ formulaci SINDy problému (1.4) funguje STLS relativně dobře [38]. Nicméně jelikož se jedná o „samostatnou metodu“, bez přesně vyjádřené ztrátové funkce, tak je ji potřeba přeformulovat, změníme-li tvar SINDy metody (například přidáním dalších omezení).

Sparse Relaxed Regularized Regression (SR3)

Jak již bylo zmíněno na začátku této kapitoly, dalším algoritmem, kterým lze nalézt šetrný model, je metoda SR3. Tato metoda, se kterou přišla Dr. Champion et al. v [38], se v jistém smyslu snaží být kompromisem mezi LASSO regresí a STLS.

SR3 upravuje optimalizační problém SINDy metody (2.5) tak, aby byl robustnější vůči odlehklým pozorováním a používal nelineární parametrický odhad [38, 47]

$$\min_{\Xi, \mathbf{W}} \frac{1}{2} \|\dot{\mathbf{X}} - \Theta(\mathbf{X})\Xi\|_2^2 + \mu R(\mathbf{W}) + \frac{1}{2\nu} \|\Xi - \mathbf{W}\|_2^2, \quad (2.7)$$

kde $\mathbf{W}$ je pomocná proměnná. Tento problém můžeme řešit pomocí příbuzné metody k ADMM, viz algoritmus 8.

```
function SR3(X,  $\dot{X}$ ,  $\theta$ ;  $\epsilon$ , W, R, v,  $\mu$ )
    k = 0
     $\epsilon$  = 2 *  $\epsilon$ 
     $\Xi$  = similar(W)
    while  $\epsilon$  >  $\epsilon$ 
        k += 1
        # Find the minimizing  $\Xi_k$  (usually via OLS or similar)
         $\Xi_k$  = argmin(1/2 * norm( $\dot{X}$  -  $\theta(X)$  *  $\Xi$ )^2 + 1/(2*v) *
        norm( $\Xi$  - W)^2 for  $\Xi$  in  $\mathbb{R}(\text{size}(W))$ )
        # Update the auxiliary variable
         $W_k$  = prox( $\Xi_k$ ,  $\mu*v*R$ )
        # Recalculate the error  $\epsilon$ 
         $\epsilon$  = norm( $W_k$  - W) / v
        W =  $W_k$ ;  $\Xi$  =  $\Xi_k$ 
    end
    return  $\Xi$ 
end
```

Algoritmus 8: Schématické znázornění algoritmu řešícího SR3

Dynamic Sequentially Thresholded Least Squares (DSTLS)

Poslední optimalizační metoda, kterou si představíme, je motivovaná problémem s rozdílnými velikostmi parametrů v jednotlivých rovnicích. Vezměme si například FitzHughův–Nagumův model (2.6)

$$\begin{aligned}\dot{V}(V, W) &= 0.8 - W + V - \frac{1}{3}V^3, \\ \dot{W}(V, W) &= 0.056 - 0.064 \cdot W + 0.08 \cdot V.\end{aligned}$$

Tento model má v první rovnici parametry v řádu desetin, ale v druhé rovnici v řádu setin. Pokud se tedy pokoušíme nalézt tento model pomocí SINDy a STLS metody (viz (1.20)), narazíme na problém s volbou parametru τ .

Zvolíme-li τ velké, např. $\tau \geq 0.1$, v druhé rovnici nenajdeme žádné vhodné členy (tedy $\dot{W}$ bude model aproximovat nulou). Naopak při volbě τ malého, např. $\tau \leq 0.03$, sice nalezneme správně dynamiku v proměnné W , ale v rovnici pro proměnnou V zbude velké množství členů s malým koeficientem.

Proto navrhujeme upravit metodu STLS tak, že v každé rovnici si metoda omezovací hranici upravuje podle již nalezených parametrů této rovnice. Jelikož se jedná o více „proměnlivou“ modifikaci STLS metody, budeme ji dále nazývat *Dynamic Sequentially Thresholded Least Squares* (DSTLS). Mimo rámec této práce usilujeme o publikaci metody DSTLS.

Jádrem naší úpravy je předpoklad, že se koeficienty v každé z rovnic od sebe výrazně neliší. Tedy místo vylučování koeficientů menších než hranice τ metoda *DSTLS* vyloučí koeficienty menší než τ -násobek největšího koeficientu. Matematicky zapsáno se jedná o úlohu

$$\min_{\xi} \frac{1}{2} \|\mathbf{Y} - \mathbf{X}\xi\|_2^2 \quad (2.8)$$

za podmínky $\forall j \in \{1, \dots, n\} : |\xi_j| > \tau \cdot \max \{|\xi| : \xi \in \xi\},$

kde uvažujeme $\tau \in (0, 1)$. Tato optimalizační metoda je ukázkově implementována v algoritmu 9. Dále na obrázcích 2.3 a 2.4 můžeme porovnat její chování oproti předešlým metodám.

```
Base.@kwdef struct DSTLS <: Optimizer
    τ = 0.1
end

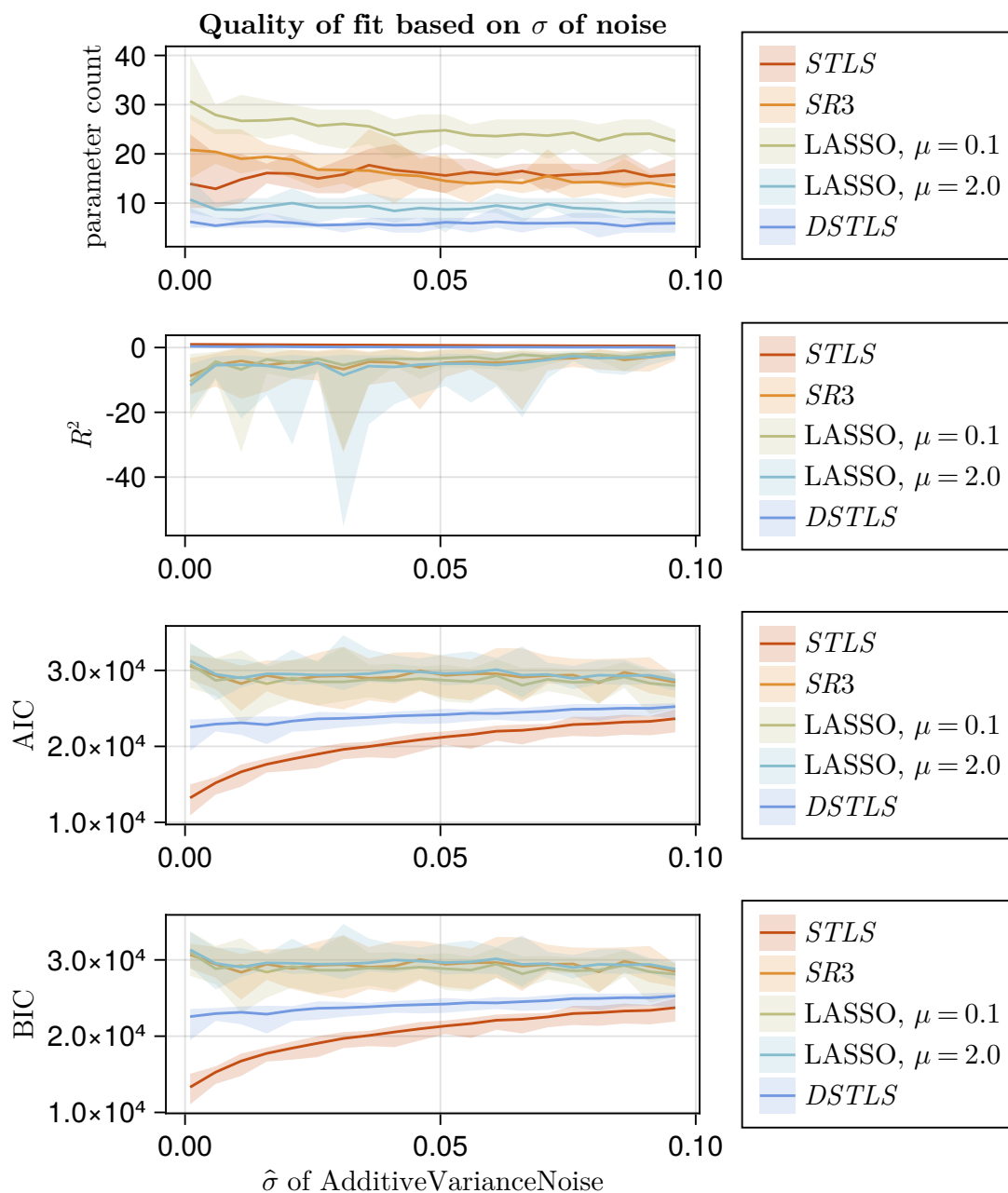
function (optimizer::DSTLS)(θ, Ẋ; max_iter=10, kwargs...)
    vars = variables(Ẋ; kwargs...)
    Ξ = θ \ Ẋ

    for _ in 1:max_iter
        for variable in 1:vars
            ξ_h = maximum(abs.(Ξ[:, variable]))
            small_indices = @. abs(Ξ[:, variable]) <
                (optimizer.τ * ξ_h)
            Ξ[small_indices, variable] .= 0

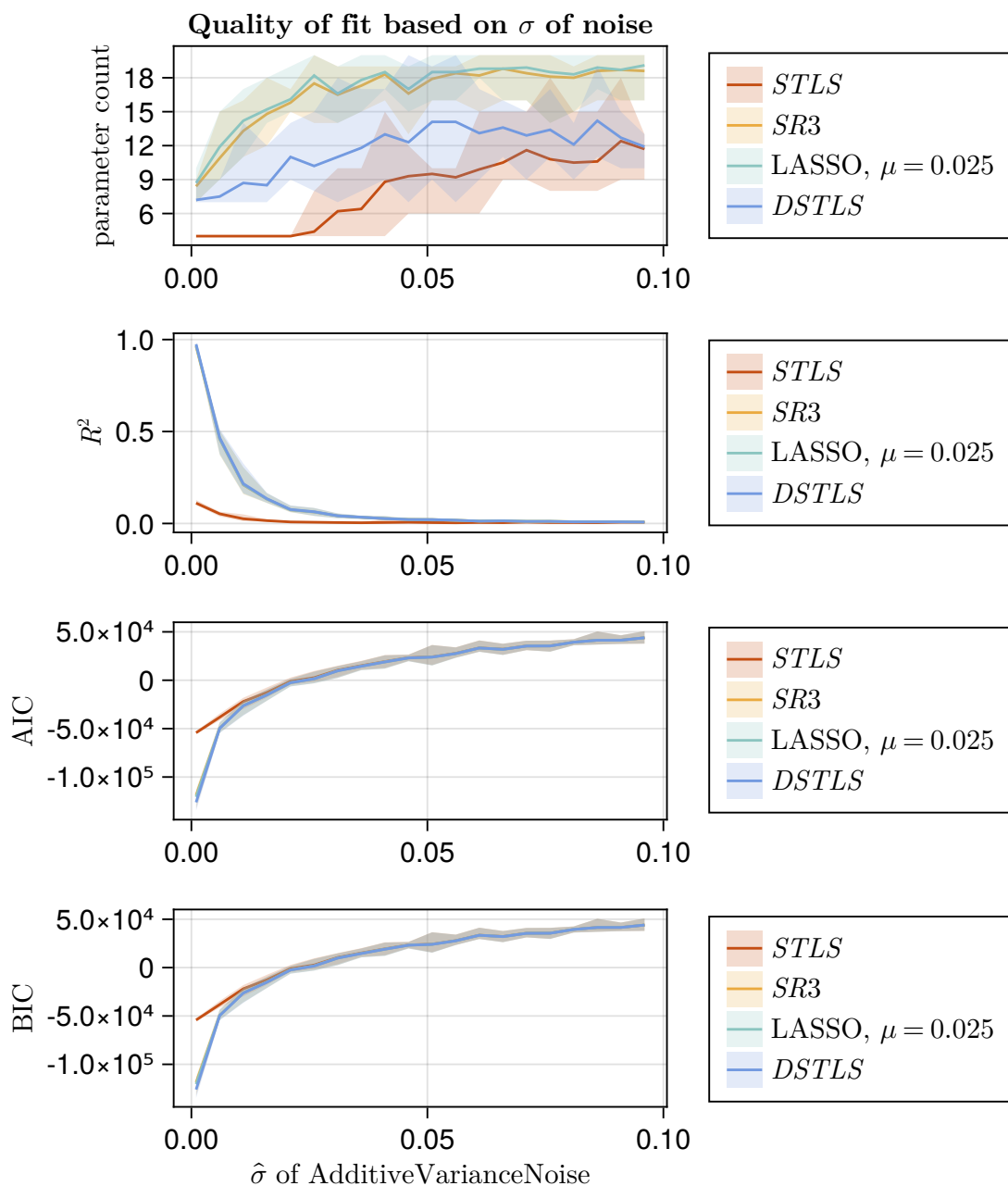
            big_indices_in = @. !small_indices
            Ξ[big_indices_in, variable] .= θ[:, big_indices_in]
        \ Ẋ[:, variable]
        end
    end

    return Ξ
end
```

Algoritmus 9: Ukázková implementace *DSTLS* metody



Obrázek 2.3: Vliv velikosti rozptylu šumu na chování SINDy metody na Lorenzově modelu ([zdrojový kód](#))



Obrázek 2.4: Vliv velikosti rozptylu šumu na chování SINDy metody na FitzHughově–Nagumově modelu ([zdrojový kód](#))

Kapitola 3

Simulační studie

V této kapitole se zaměříme na aplikace SINDy metody na vybrané problémy, zvláště pak na modely neuronů. Podíváme se podrobněji na chování SINDy metody, různé diagnostiky kvality konvergence apod.

3.1 FitzHughův–Nagumův model

První a nejjednodušší model, který jsme si navíc již ukázali v předchozích kapitolách, je FitzHughův–Nagumův model neuronu. Jedná se o 2-rozměrné zjednodušení Hodgkinova–Huxleyho modelu vzniku a šíření akčního potenciálu [48]. Hodgkinův–Huxleyho model (HH model, viz 3.3) byl vytvořen na základě měření velkého neuronu olihně [49] a je tvořen 4 proměnnými. FitzHughův–Nagumův model je z něj odvozen pomocí lineární kombinace původních proměnných, viz sekce 3.3. Podívejme se nyní na biologickou podstatu zkoumaného systému.

Každá buňka má na svém okraji **buněčnou membránu**, která (většinou) dobře propouští draselné kationty K^+ , ale špatně kationty sodíku Na^+ . Díky této vlastnosti buněčné membrány obecně u všech buněk dochází k jevu, který nazýváme **membránový potenciál**. To je stav, kdy je koncentrace kladně nabitých částic (typicky iontů K^+ , které difundují ven z buňky) uvnitř buňky nižší, než vně buňky (typicky Na^+ a K^+). Jak draslíkové ionty K^+ opouští buňku, zanechávají v ní anionty Cl^- , které ale podle Coulombova zákona zpátky přitahují kladně nabitě částice směrem do buňky (avšak opět jen kationty K^+ mohou projít přes buněčnou membránu). Tento nepoměr kladných iontů na obou stranách buněčné membrány vytváří elektrický potenciál, který se stabilizuje na hladině -70 mV, což nazýváme *klidový membránový potenciál*.

Některé buňky, jako například neurony, jsou schopné kontrolovat propustnost své buněčné membrány. **Akčním potenciálem** nazýváme jev, při kterém dochází k lokální změně membránového potenciálu. Tento jev je způsoben změnou koncentrace iontů na vnitřní straně buňky. Je-li tato změna pouze mírná, regulační mechanismy ji vyrovnají na původní hladinu klidového membránového potenciálu a k popisovanému jevu nedochází. Naopak za přítomnosti dostatečně výrazné změny dojde ke zmiňovanému akčnímu potenciálu.

Při tomto procesu buňka otevře Na^+ -propustné kanálky v buněčné membráně,

kteřé jsou normálně zavřené, a Na^+ ionty se díky Coulombovu zákonu a difúzi nahrnou dovnitř buňky. Buňka se takto stane kladně nabitou a kationty K^+ , které uvnitř držela Coulombova síla, se dostanou ven z buňky. V této chvíli je na okamžik hodnota membránového potenciálu ještě nižší než klidový membránový potenciál, čemuž říkáme *hyperpolarizace*. Celý akční potenciál končí navrácením membránového potenciálu na klidovou hladinu. Tato fáze navrácení na klidový stav má dvě části – absolutní, při které nemůže dojít k akčnímu potenciálu, a relativní, při které je výše změny membránového potenciálu nutná na spuštění akčního potenciálu vyšší než za normálního stavu.

Jak již bylo zmíněno, akční potenciál je lokální jev, avšak nastane-li v jednom místě, „řetězovou reakcí“ se začne šířit po buněčné membráně. Díky výše popsané klidové fázi typicky nedochází k nekonečnému koloběhu akčních potenciálů, při němž by akční potenciál okolo místa prvotního akčního potenciálu vyvolal v počátku opět akční potenciál.

Neurony se řídí tzv. pravidlem „všechno nebo nic“ (*all-or-nothing law*), tedy jakmile dostanou dostatečně silný vzruch, vyšlou signál přes **akční potenciál**. Tento signál je navíc pokaždé prakticky stejný a po celé buňce neměnný – jinak řečeno, při šíření nedochází k deformaci signálu. Tento mechanismus používají neurony na komunikaci mezi sebou¹.

Z tohoto mimo jiné plyne, že neuron je schopen poznat pouze to, zda k němu zpráva (akční potenciál sousedního neuronu) dorazila, kolik jich momentálně bylo, či od kterého neuronu přišla. Avšak všechny zprávy od daného neuronu jsou *de facto* totožné [49].

Zjednodušený model akčního potenciálu, odvozený z HH modelu, byl představen v [51], kde jej autor nazývá *Bonhoefferův–Van der Polův model* (BVP model), avšak my jej budeme nazývat *FitzHughův–Nagumův model* (FHN model).

Jak již bylo uvedeno dříve, FHN model má tvar (1.2)

$$\begin{aligned}\dot{V} &= V - \frac{V^3}{3} - W + i_e, \\ \dot{W} &= a \cdot (b \cdot V - c \cdot W + d),\end{aligned}$$

kde je význam vybraných proměnných a parametrů následující:

V zastupuje hodnotu membránového potenciálu napříč buněčnou stěnou [49],

W tzv. *obnovovací proměnná* (anglicky *recovery variable*), která popisuje zotavení neuronu po akčním potenciálu,

i_e velikost excitačního (*budícího*) proudu,

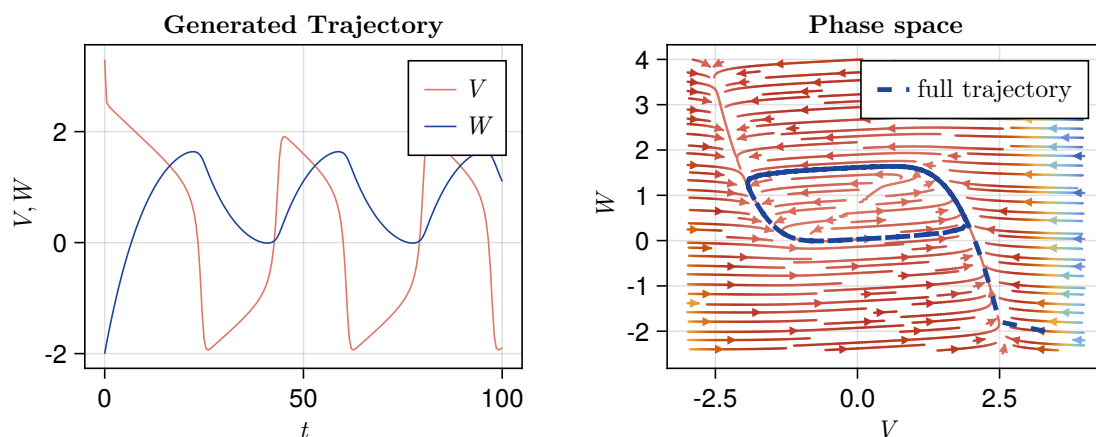
a, b, c, d parametry popisující vlastnosti neuronu a prostředí.

V této práci, nebude-li řečeno jinak, budeme volit hodnoty parametrů

$$\begin{aligned}a &= 0.08, b = 1, c = 0.8, d = 0.7, \\ i_e &= 0.8,\end{aligned}$$

¹Buněčná membrána může obsahovat více kanálků, než jsme zde pro jednoduchost popsali – viz [50].

s počáteční podmínkou $[V, W] = [3.3, -2]$ na časovém intervalu $(0, 100)$. Systém diferenciálních rovnic daný modelem FHN (1.2) řešíme pomocí Rosenbrock23² metody na numerické řešení obyčejných diferenciálních rovnic. Zobrazenou trajektorii systém za těchto podmínek můžeme vidět na obrázcích 3.1.



(a) Vygenerovaná trajektorie FHN modelu (b) Fázový portrét FHN modelu s vyznačenou trajektorií

Obrázek 3.1: Přehled vygenerované FHN trajektorie ([zdrojový kód](#))

Jak je vidět na obrázcích 3.1, při těchto parametrech docházím k opakovanému akčnímu potenciálu (na obrázku 3.1b je to vystiženo limitním cyklem, viz definice 9).

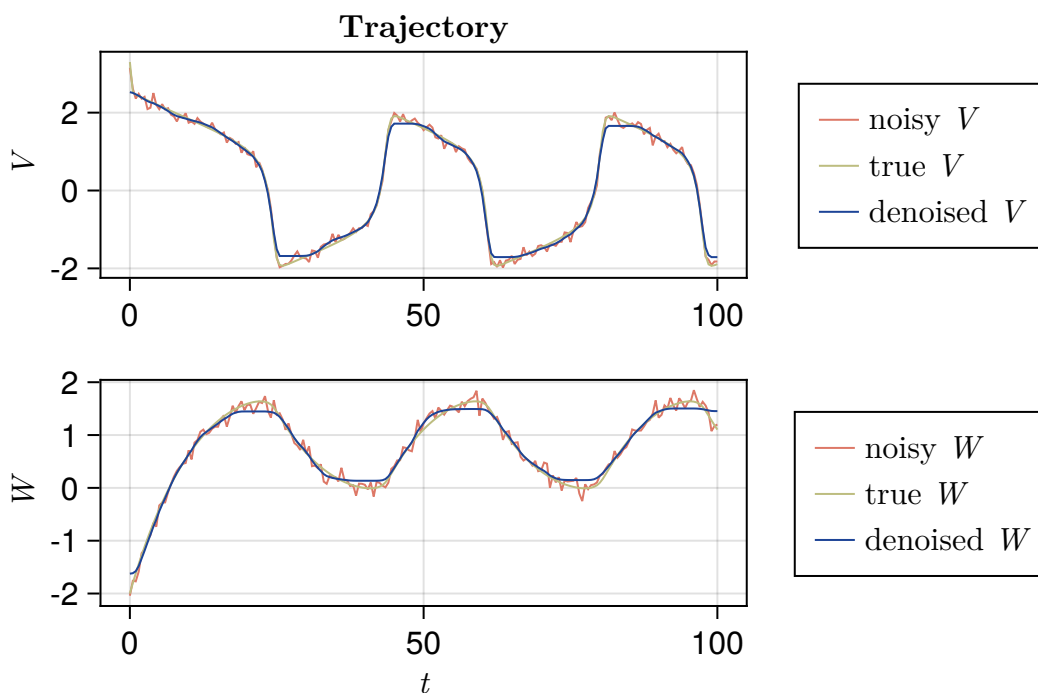
Takto získaná data použijeme na vstupu SINDy metody, přičemž předpokládáme, že známe měření stavových proměnných V, W , která jsou ovlivněna *aditivním šumem* s $\mu = 0, \sigma = 0.1$ pocházejícím z normálního rozdělení (viz sekce 1.4.1), a navíc **neznáme** jejich derivace.

Jak jsme si ukázali v sekci 1.4.2, na odstranění šumu z dat použijeme metodu totální variace. Srovnání původní trajektorie, trajektorie ovlivněné šumem a po odstranění šumu můžeme vidět na obrázku 3.2.

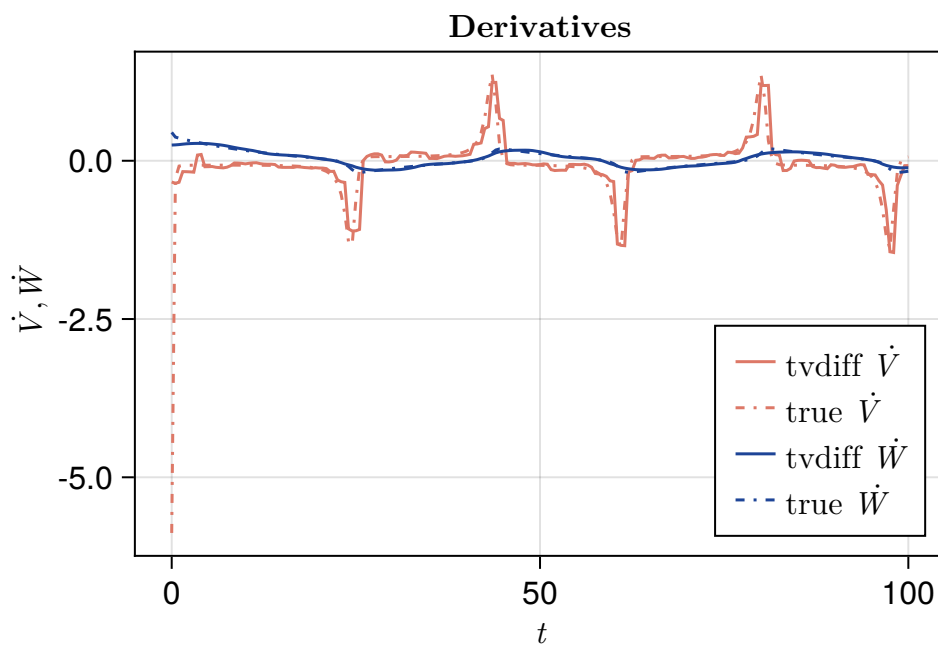
Obdobný proces provedeme i pro získání derivací ze zašuměných dat pomocí *numerického derivování regularizovaného totální variací* (viz sekce 1.4.3). Srovnání skutečných derivací a derivací nalezených touto metodou je zobrazeno na obrázku 3.3.

Nyní potřebujeme zvolit knihovnu kandidátních funkcí pro SINDy metodu. Tentokrát zvolme polynomiální knihovnu proměnných V, W nejvýše 4. stupně. Jakožto optimalizační metodu použijeme námi navrhovanou metodu *DSTLS*,

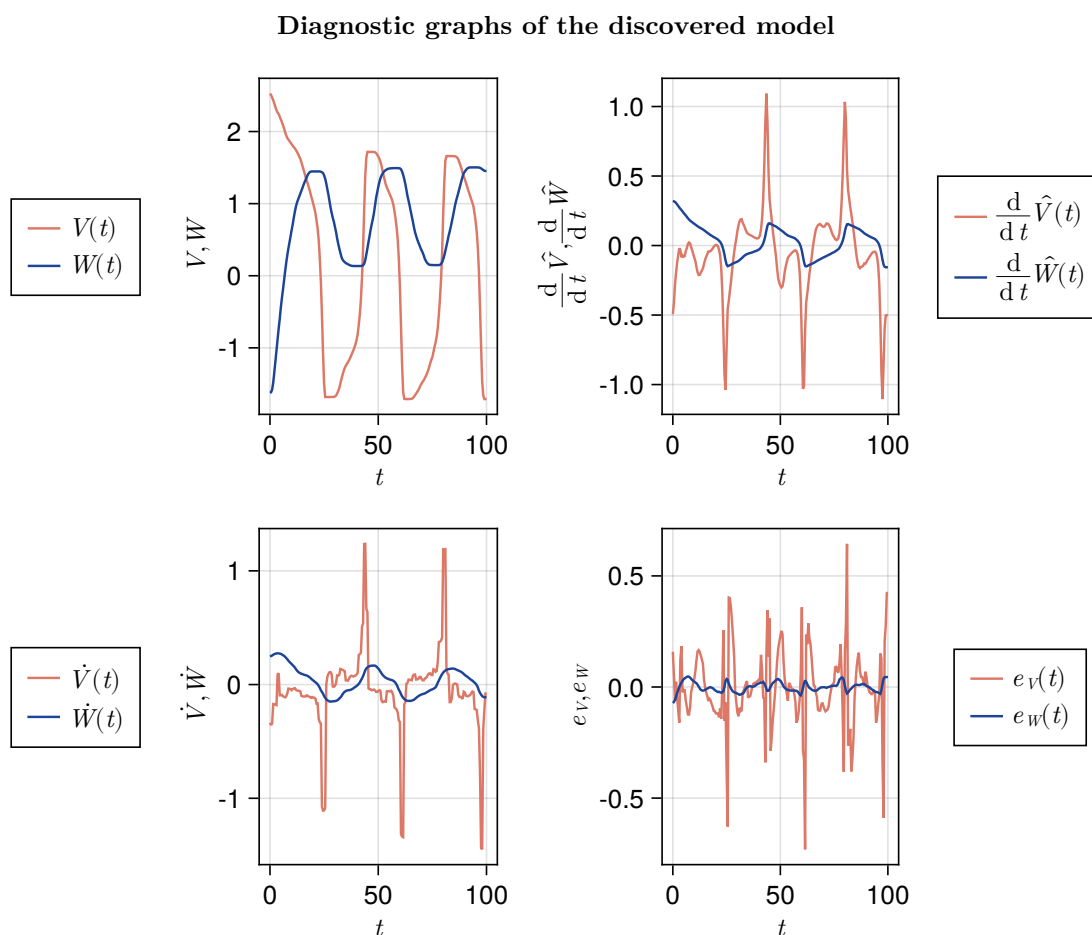
²Alternativně na řešení diferenciální rovnice můžeme použít např. `Tsit5`, přičemž doporučená metoda dobře funguje i na *tuhé* problémy.



Obrázek 3.2: Srovnání trajektorií po procesu odstranění šumu ([zdrojový kód](#))



Obrázek 3.3: Srovnání skutečné derivace a derivace získané ze zašuměných dat ([zdrojový kód](#))



Obrázek 3.4: Diagnostické grafy naučeného modelu ([zdrojový kód](#))

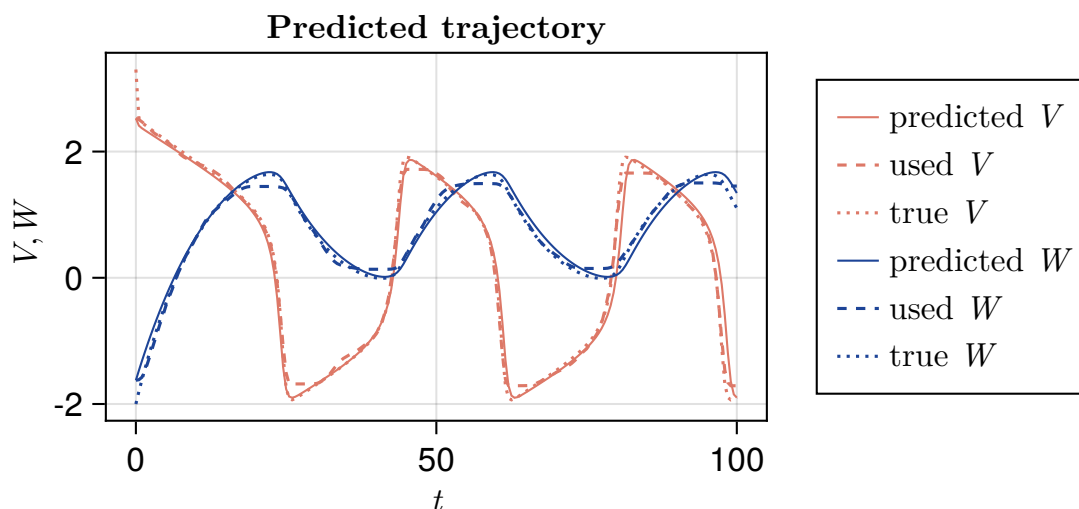
viz 2.4.4, s $\tau = 0.15$. Celkem pomocí SINDy metody dostaneme naučený systém

$$\begin{aligned}\frac{d}{dt}\hat{V} &= 0.7 + 0.83 \cdot \hat{V} - 0.85 \cdot \hat{W} - 0.29 \cdot \hat{V}^3, \\ \frac{d}{dt}\hat{W} &= 0.05 + 0.08 \cdot \hat{V} - 0.05 \cdot \hat{W}.\end{aligned}$$

Diagnostické grafy naučeného modelu lze vidět na obrázku 3.4. Vlevo nahoře je zobrazená trajektorie použitá na nalezení modelu, vlevo dole pak použité derivace. Vpravo nahoře jsou derivace odhadnuté naučeným modelem a vlevo dole rozdíl původních a nových derivací v každém časovém okamžiku.

V této chvíli už máme nalezený model a pokud jsme s ním spokojeni, tak naše práce končí. Jako poslední „měřítko kvality“ nalezeného modelu můžeme použít srovnání originální trajektorie a trajektorie simulované pomocí nového modelu ze stejného počátečního bodu, viz obrázek 3.5.

Pro úplnost ještě doplňme, že ačkoliv se nám v tomto případě podařilo model nalézt relativně úspěšně, změníme-li některé „parametry“, např. velikost knihovny



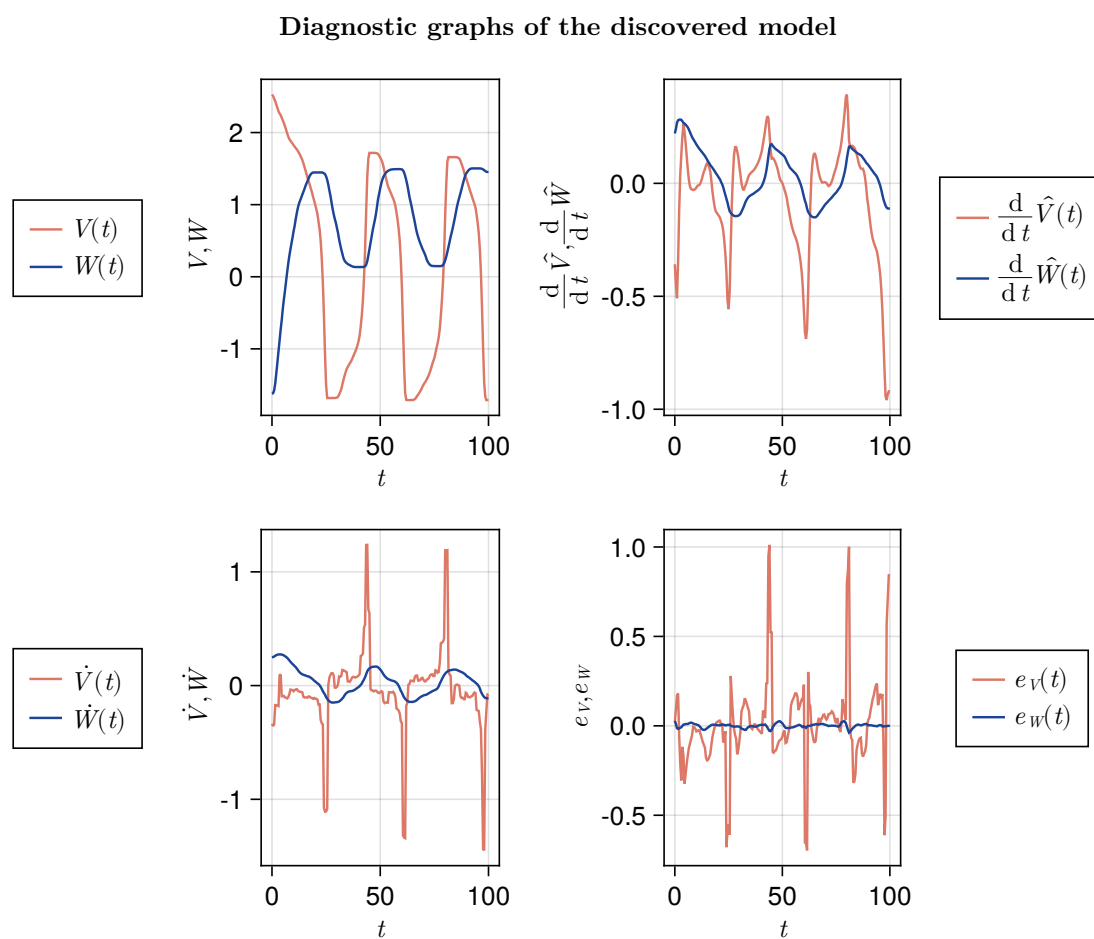
Obrázek 3.5: Srovnání původní a nově simulované trajektorie ([zdrojový kód](#))

či optimalizační metodu, výsledky se mohou velice lišit. Jako příklad si uveďme stejnou úlohu, avšak s polynomiální knihovnou nejvýše 5. stupně. Výsledkem takové úlohy je systém

$$\begin{aligned} \frac{d}{dt} \hat{V} &= 2.01 \cdot \hat{W} + 0.65 \cdot \hat{V} \cdot \hat{W} + 0.76 \cdot \hat{V} \cdot \hat{W}^3 + 0.58 \cdot \hat{V}^2 \cdot \hat{W}^2 \\ &\quad - 1.07 \cdot \hat{W}^3 - 1.44 \cdot \hat{V} \cdot \hat{W}^2 - 0.88 \cdot \hat{W} \cdot \hat{V}^2, \\ \frac{d}{dt} \hat{W} &= 0.06 + 0.09 \cdot \hat{V} + 0.02 \cdot \hat{W}^4 + 0.06 \cdot \hat{V} \cdot \hat{W}^2 + 0.04 \cdot \hat{W} \cdot \hat{V}^3 \\ &\quad + 0.04 \cdot \hat{V}^2 \cdot \hat{W}^2 - 0.05 \cdot \hat{W}^2 - 0.02 \cdot \hat{W}^3 - 0.11 \cdot \hat{V} \cdot \hat{W} - 0.04 \cdot \hat{W} \cdot \hat{V}^2 \\ &\quad - 0.02 \cdot \hat{V}^3 \cdot \hat{W}^2, \end{aligned}$$

kterého diagnostické grafy lze vidět na obrázku 3.6. Na první pohled je patrné, že nalezený model je výrazně horší kvality než v předchozím případě. Pokud bychom se snažili, mohli bychom nejspíše zlepšit kvalitu tohoto model změnou parametru τ u optimalizační metody *DSTLS*, či bychom mohli optimalizovat pomocí jiné metody, např. *STLS* nebo *SR3*, což by značně ovlivnilo nalezený model.

Také si všimněme, že ani z dat nalezená derivace neodpovídá pravdě tak, jak bychom si možná přáli. I v tomto případě by stálo za to buď změnit parametry použité metody nebo získat tuto derivaci jiným způsobem.

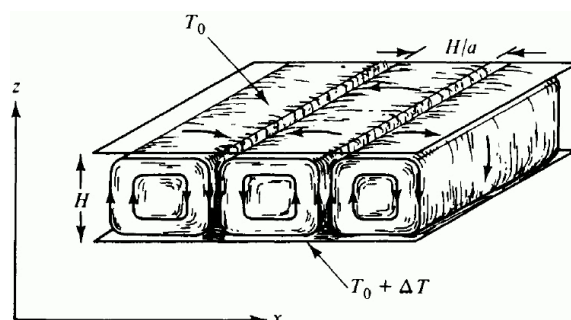


Obrázek 3.6: Diagnostické grafy naučeného modelu s polynomiální knihovnou nejvýše 5. stupně ([zdrojový kód](#))

3.2 Lorenzův model

Ačkoliv se nejedná o model neuronu, z našeho pohledu je pro celkové pochopení chování SINDy metody Lorenzův model nezbytný, neboť u něj pro vhodně zvolené parametry dochází k *deterministickému chaosu*. Ačkoliv lze najít modely neuronu, které tak vykazují deterministický chaos, jmenovitě Hindmarshův–Roseův model, případně i FitzHughův–Nagumův model s externím buzením zadáním vhodnou funkcí, žádný z těchto modelů není tak „jednoduchý“ a tedy ilustrativní jako Lorenzův model. Proto jsme se rozhodli ukázat chování SINDy a deterministického chaosu právě na Lorenzově modelu.

V roce 1963 meteorolog a matematik Edward Lorenz sestavil zjednodušený model proudění v atmosféře [52], když se svými kolegy, jmenovitě s Ellen Fetterovou a Dr. Barry Saltzmanem, zkoumali dlouhodobé předpovědi počasí – přesněji otázku, proč jsou tak nepřesné. Lorenz pro tyto potřeby sestavil model, kterým modeloval konvekční proudění v případě, že uvažujeme tekutou (nejčastěji vzduchovou) vrstvu, jež je zahřívána zespodu a chlazená shora. Tím se v dolní části tekutina ohřívá, stává se méně hustou než tekutina nad ní a začíná proudit nahoru. Naopak chlazená tekutina v horní části se stává hustější než tekutina pod ní a tlačí se dolů. Takto vzniká v tekuté vrstvě proudění tvořící tzv. *konvekční buňky*, což se Lorenz snažil popsat tímto modelem, viz obrázek 3.7.



Obrázek 3.7: Schematické znázornění modelované situace [53]

Na obrázku 3.7 parametr H určuje výšku tekuté vrstvy, T_0 je teplota na vrchu popisované vrstvy a ΔT značí rozdíl teplot mezi spodní a vrchní vrstvou. V neposlední řadě H/a udává „šířku buňky konvekčního proudění“.

Lorenzův model je dán 3 diferenciálními rovnicemi

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z,\end{aligned}\tag{3.1}$$

kde význam proměnných a parametrů je následující [54, 53]:

x rychlost rotace konvekčního proudění,

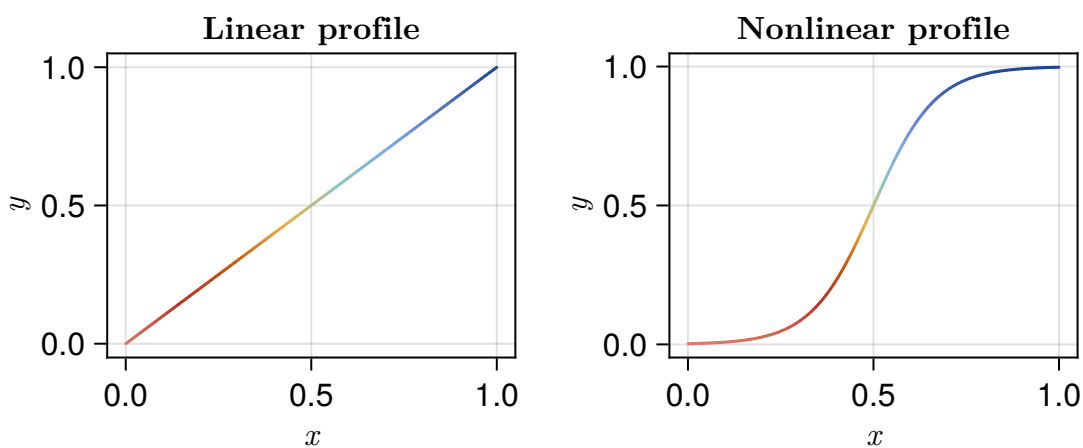
y rozdíl teploty stoupajícího a klesajícího proudu tekutiny,

z odchylka vertikálního gradientu (*profilu*) teplot od linearity, viz obrázek 3.8,

$\sigma = \frac{\nu}{\kappa}$ Prandtlovo číslo (ν odpovídá kinematické viskozitě a κ součiniteli tepelné vodivosti),

$\rho = \frac{R}{R_c}$ podíl Rayleighova čísla R a kritického Rayleighova čísla R_c , viz [53],

$\beta = \frac{4}{1+a^2}$ koeficient rozměrů buněk konvekčního proudění – vztahuje se k poměru šířky a délky jedné konvekční buňky.



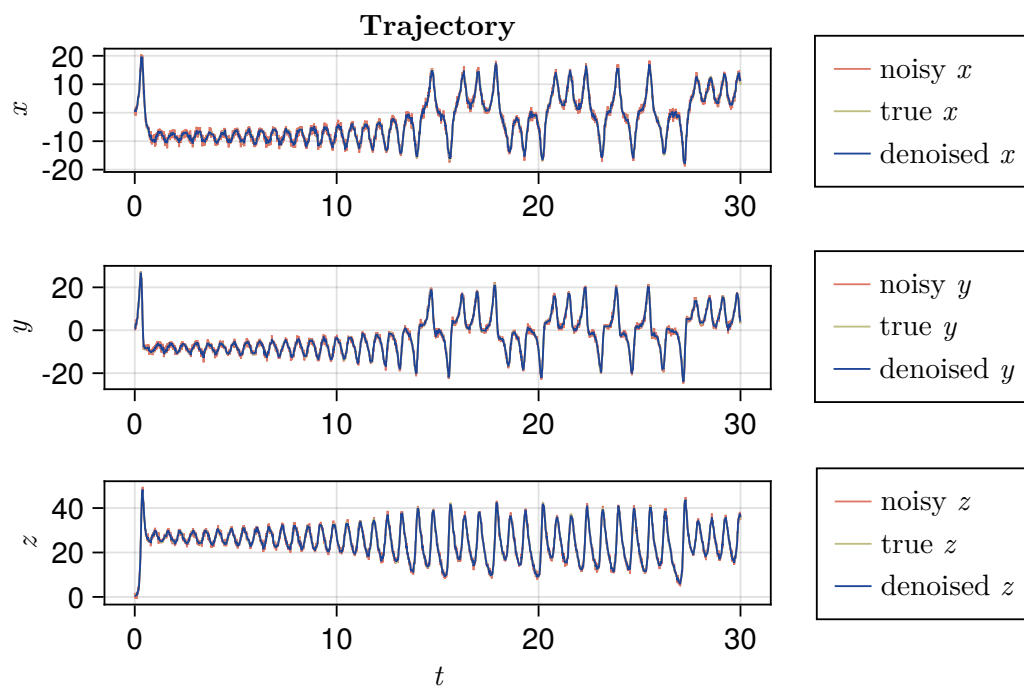
Obrázek 3.8: Znázornění lineárního a nelineárního teplotního profilu ([zdrojový kód](#))

V našem případě budeme nejčastěji volit parametry stejné jako v originální publikaci [52], tj. $\sigma = 10, \rho = 28, \beta = \frac{8}{3}$. Při těchto hodnotách parametrů dochází k deterministickému chaosu, což chceme společně se SINDy zkoumat. Dále v této sekci budeme předpokládat, že Lorenzův systém simulujeme na časovém intervalu $(0, 30)$ a není-li řečeno jinak, uvažujeme výchozí bod $[1, 1, 1]$.

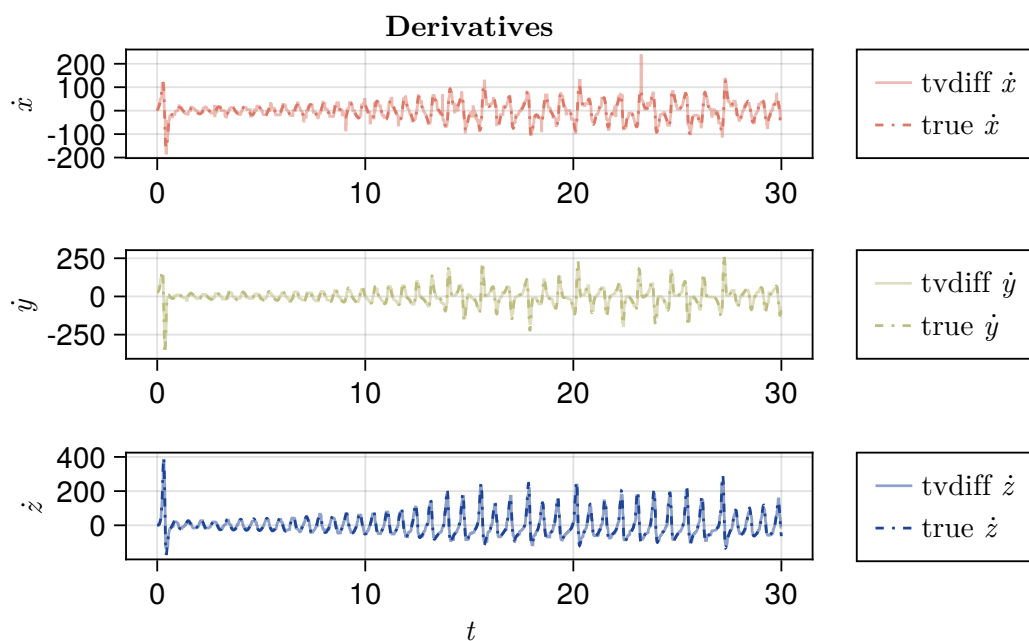
Takto definovaný problém systému obyčejných diferenciálních rovnic opět řešíme numericky pomocí metody `Rosenbrock23` a ukládáme si trajektorii po časových krocích $\Delta t = 0.01$. Stejně jako v předchozí sekci, také zde předpokládáme, že **nemáme** měření derivací a data jsou zatížena aditivním šumem $\mathcal{N}_i \stackrel{\text{iid}}{\sim} N(0, 1)$.

Dále je zřejmě potřeba pokusit se odstranit šum z dat a spočítat derivace z původních dat ovlivněných šumem. I tentokrát použijeme *metodu totální variace*, viz obrázek 3.9, na vyhlazení dat a *numerické derivování regularizované totální variací*, viz obrázek 3.10.

Jako knihovnu kandidátních prvků použijeme polynomiální knihovnu nejvýše 3. stupně – knihovna polynomů 2. stupně by byla nejmenší možná, tedy polynomy 3. stupně tvoří dostatečně, ale ne příliš, velký prostor „možných pravých stran“ pro naši ukázkou. Volba optimalizační metody se u tohoto modelu ukáže neméně



Obrázek 3.9: Srovnání trajektorií po procesu odstranění šumu ([zdrojový kód](#))



Obrázek 3.10: Srovnání skutečné derivace a derivace získané ze zašuměných dat ([zdrojový kód](#))

důležitá. První tedy zvolme metodu *STLS* s $\tau = 0.4$ (tj. kovariáty s koeficienty větší než 0.4 jsou pokaždé vyřazeny z regrese). S těmito předpoklady dostaneme pomocí *SINDy* metody nalezený model

$$\begin{aligned}\frac{d}{dt}\hat{x} &= 0.86 + 9.3 \cdot \hat{y} - 9.08 \cdot \hat{x}, \\ \frac{d}{dt}\hat{y} &= 18.27 \cdot \hat{y} - 7.14 \cdot \hat{x} - 0.49 \cdot \hat{y} \cdot \hat{z}, \\ \frac{d}{dt}\hat{z} &= 0.93 \cdot \hat{x} \cdot \hat{y} - 11.4 - 2.02 \cdot \hat{z}.\end{aligned}\tag{3.2}$$

Všimněme si zde chybějící „interakce“ ve druhé rovnici mezi členy $\hat{x}$ a $\hat{z}$, tj. chybějící $\hat{x} \cdot \hat{z}$, a naopak špatně identifikované interakce $\hat{y} \cdot \hat{z}$. Také velikosti koeficientů u $\hat{x}$ a $\hat{y}$ neodpovídají původním hodnotám. Ve stejném duchu se nese i 3. rovnice, která obsahuje nadbytečný konstantní posun nezanedbatelné velikosti.

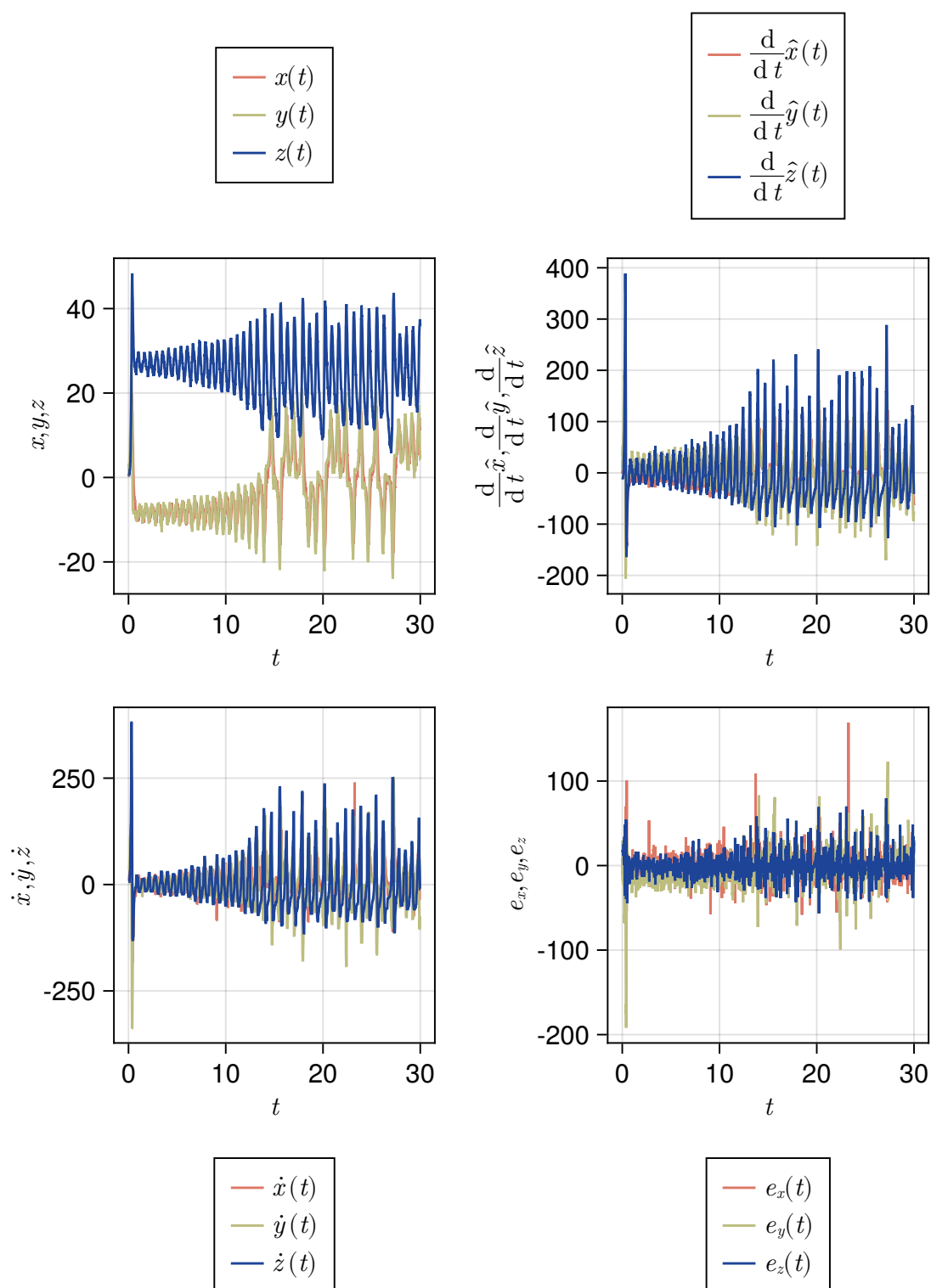
Diagnostické grafy modelu nalezeného pomocí optimalizační metody *STLS* jsou ukázány na obrázku 3.11. Zaměřme se na tomto obrázku hlavně na srovnání původních a nalezených derivací – zde se nám podařilo relativně dobře odhadnout derivace, což vidíme i na pravém dolním obrázku. Navíc jsou chyby nalezených derivací více méně konstantní na celých datech.

Pokud bychom simulovali nalezený model ze stejného počátečního bodu na stejném časovém intervalu, zjistíme, že trajektorie se velmi brzo rozcházejí, viz obrázek 3.12. Toto je mimo jiné způsobeno i deterministickým chaosem přítomným v Lorenzově modelu – deterministický chaos se pravděpodobně bude objevovat i v nalezeném modelu, neboť ten se snaží aproximovat původní Lorenzův model. Povšimněme si, že v proměnných x a y u nově nalezeného modelu nevidíme tzv. *střídání křídel* Lorenzova atraktoru (anglicky *lobe-switching*), které je patrné na původní trajektorii.

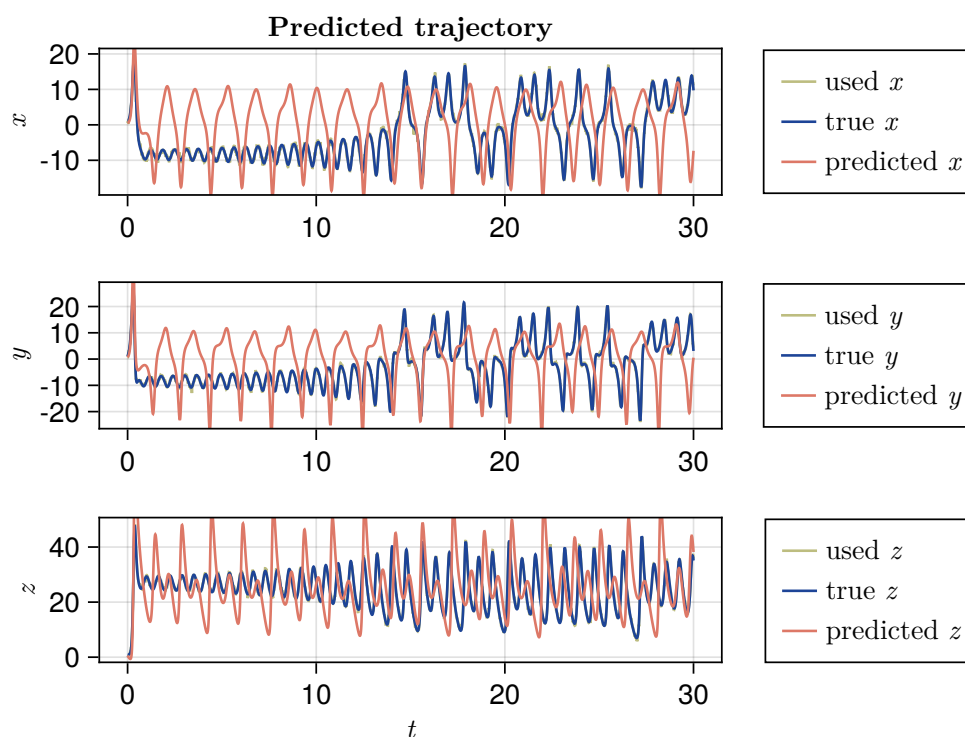
Ačkoliv přesná predikce trajektorie je užitečná, tak v tomto případě vidíme, že zvláště z dlouhodobého hlediska je to nemožné. Nicméně velice užitečnou informací může být atraktor, ke kterému jsou ostatní trajektorie přitahovány. Pokud se nám podaří správně identifikovat atraktor, tak ačkoliv nejsme schopni přesně předpovědět trajektorii, jsme schopni říct, v jaké množině bude za dostatečně dlouhou dobu ležet. Srovnání původního a nového atraktoru můžeme vidět na obrázku 3.13. Zde si můžeme všimnout, že nalezený atraktor neodpovídá Lorenzovu atraktoru tak, jak bychom si mohli přát.

Z předcházející analýzy a obrázku můžeme usoudit, že ačkoliv pěkně odpovídají nalezené derivace, celková kvalita nalezeného modelu nemusí být dostatečná. Proto se pokusme tento model zlepšit použitím námi navrhované metody *DSTLS* s $\tau = 0.02$ (stejně jako v minulém případě, i tentokrát jsme „optimální“ hodnotu τ našli metodou pokus–omyl). Nalezený *SINDy* model s touto optimalizační me-

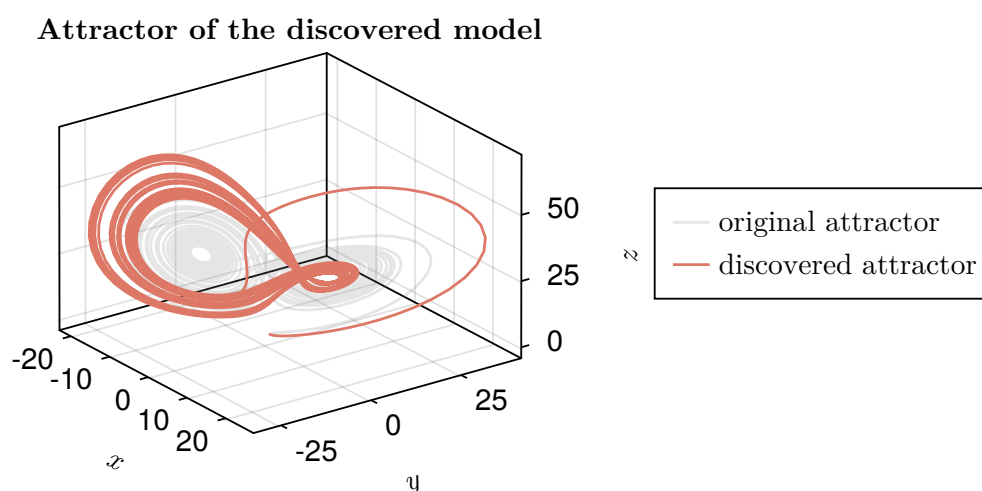
Diagnostic graphs of the discovered model



Obrázek 3.11: Diagnostické grafy naučeného modelu s optimalizační metodou *STLS* pro $\tau = 0.4$ ([zdrojový kód](#))



Obrázek 3.12: Srovnání původní a nově simulované trajektorie s optimalizační metodou STLS pro $\tau = 0.4$ ([zdrojový kód](#))



Obrázek 3.13: Srovnání atraktoru pro původní a nalezený systém s optimalizační metodou STLS pro $\tau = 0.4$ ([zdrojový kód](#))

todou bude mít tvar

$$\begin{aligned}\frac{d}{dt}\hat{x} &= 0.86 + 9.3 \cdot \hat{y} - 9.08 \cdot \hat{x}, \\ \frac{d}{dt}\hat{y} &= 23.01 \cdot \hat{x} - 0.86 \cdot \hat{x} \cdot \hat{z}, \\ \frac{d}{dt}\hat{z} &= 0.14 \cdot \hat{x}^2 + 0.84 \cdot \hat{x} \cdot \hat{y} - 2.51 - 2.51 \cdot \hat{z},\end{aligned}\tag{3.3}$$

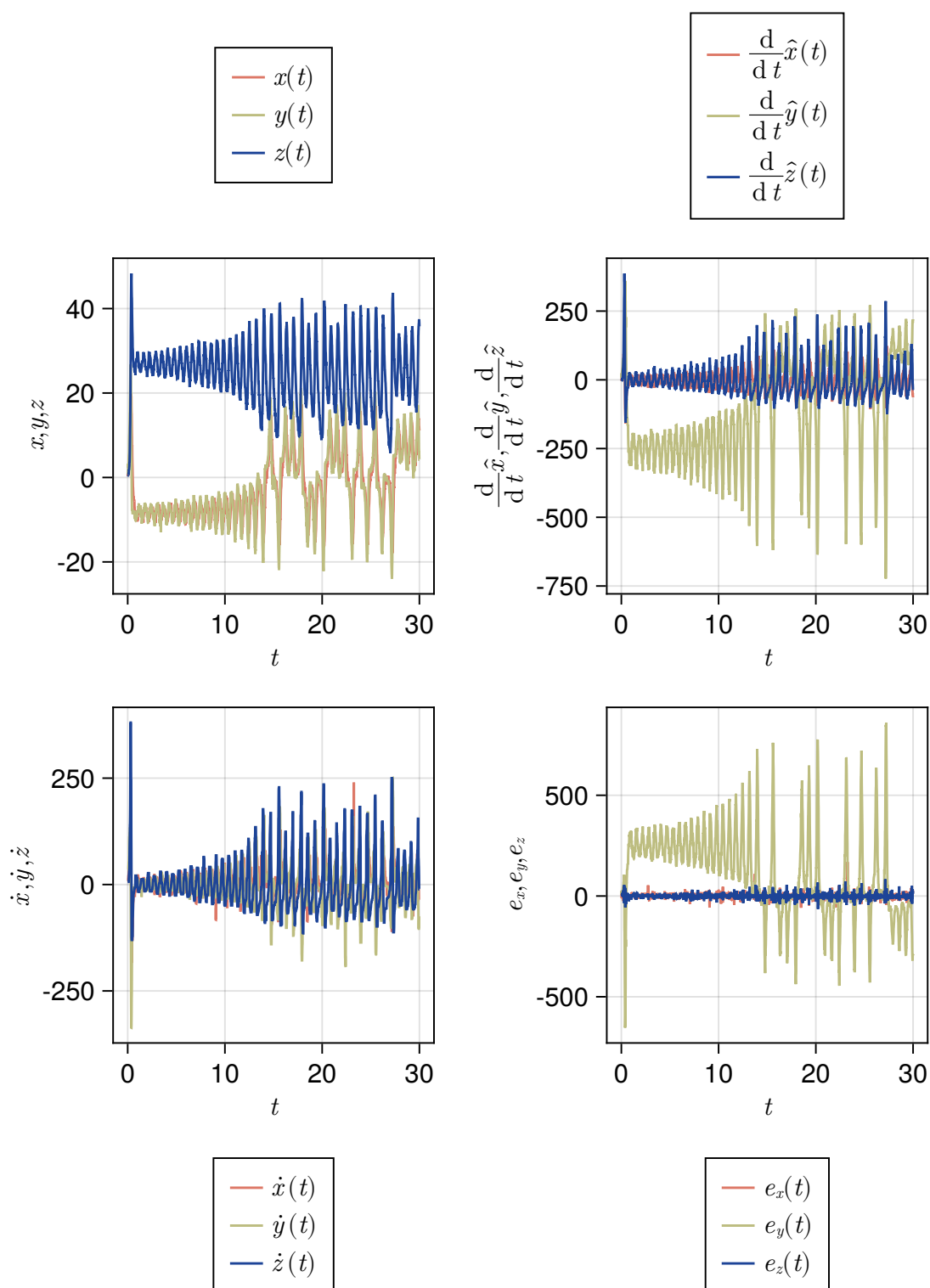
kde si povšimněme, že jsme tentokrát našli správnou interakci ve 2. rovnici i hodnotu koeficientu u členu $\hat{x}$, ačkoliv nám nyní chybí člen $\hat{y}$. Dále 1. rovnice zůstala *de facto* totožná. Ve 3. rovnici jsme chybně identifikovali člen $\hat{x}^2$, který ale má v nalezeném modelu relativně malý koeficient, a podařilo se nám zmenšit konstantní člen v této rovnici – ten se v originálním modelu také nevyskytuje. Diagnostické grafy nalezeného SINDy modelu lze vidět na obrázku 3.14, kde si můžeme všimnout výrazně horší kvality nalezené derivace $\frac{d}{dt}\hat{y}$ než u optimalizační metody STLS.

Ačkoliv by se z diagnostických grafů mohlo zdát, že nový model (3.3) je „horší“ než předchozí (3.2), zjistíme, že nemůžeme spoléhat pouze na ně. Pokud simulujeme nalezený systém (3.3) opět ze stejného počátečního bodu a na stejném časovém intervalu, dostaneme trajektorii zobrazenou na obrázku 3.15. Nejen, že tato simulovaná trajektorie nalezeného systému vykazuje *střídání křídel*, ale i y -část nové trajektorie se chová podobně jako v trajektorii Lorenzova modelu, o čemž bychom mohli kvůli chybě nalezené a původní derivace y pochybovat.

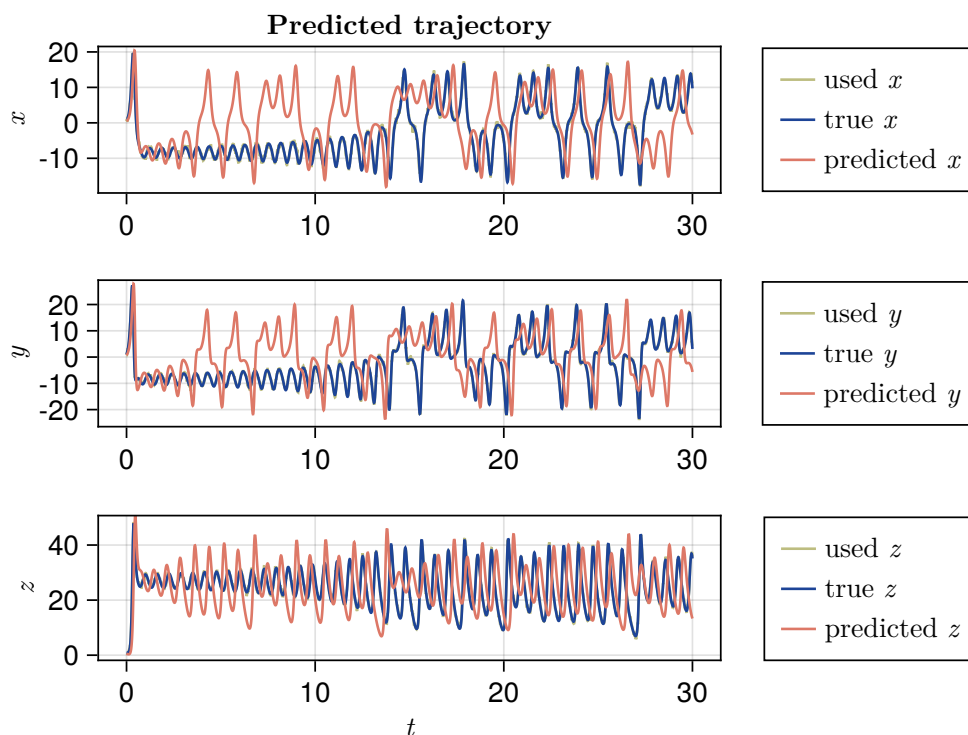
Asi nejzajímavější bude srovnání původního Lorenzova atraktoru s atraktorem pro nalezený systém. Vzpomeňme si na metodou STLS nalezený atraktor, viz obrázek 3.13, který neodpovídal originálnímu Lorenzovu atraktoru. Naopak atraktor podle metody DSTLS, viz obrázek 3.16, výrazně lépe odpovídá Lorenzovu atraktoru – a tedy by byl značně užitečnější na zevrubnou predikci trajektorie.

Ačkoliv jsme si doteď v této kapitole uvedli nanejvýše srovnání metod STLS a DSTLS, jak bylo popsáno i v sekci 2.4.4, mohli jsme použít např. LASSO regresi či SR3 metodu. Také volbu knihovny jsme prováděli víceméně „optimisticky“, kdy jsme již měli představu, jaké funkce budou tvořit pravé strany diferenciálních rovnic a zvolené knihovny nebyly příliš rozsáhlé. I přesto doufáme, že tato část poslední kapitola poskytla vhled do použití SINDy metody na vybrané problémy, spolu s tipy, či triky, které jsme při děláni této práce objevili.

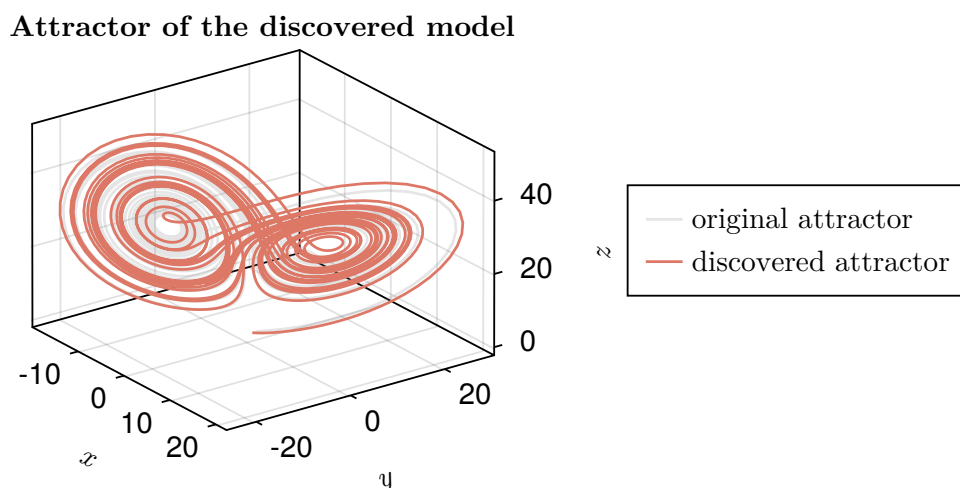
Diagnostic graphs of the discovered model



Obrázek 3.14: Diagnostické grafy naučeného modelu s optimalizační metodou DSTLS pro $\tau = 0.02$ ([zdrojový kód](#))



Obrázek 3.15: Srovnání původní a nově simulované trajektorie s optimalizační metodou *DSTLS* pro $\tau = 0.02$ ([zdrojový kód](#))



Obrázek 3.16: Srovnání atraktoru pro původní a nalezený systém s optimalizační metodou *DSTLS* pro $\tau = 0.02$ ([zdrojový kód](#))

3.3 Transformace dynamiky Hodgkinova–Huxleyho modelu na FitzHughův–Nagumův model

Následující sekce pojednává o metodách transformace Hodgkinova–Huxleyho modelu na jednodušší FitzHughův–Nagumův model. Prezentované výsledky i postupy, včetně druhů použitých transformací a metodiky optimalizace, jsou čistě autorskou prací.

Jak již bylo dříve zmíněno, FitzHughův–Nagumův model je odvozený ze složitějšího Hodgkinova–Huxleyho modelu. Vznik a biologické vysvětlení 4-rozměrného Hodgkinova–Huxleyho modelu bylo rozepsáno v sekci 3.1, proto si nyní uveďme pouze jeho tvar:

$$\begin{aligned}\dot{V} &= \frac{1}{C_m} (I_e - g_{Na}m^3h(V - V_{Na}) - g_Kn^4(V - V_K) - g_L(V - V_L)), \\ \dot{m} &= \alpha_m(V)(1 - m) - \beta_m(V)m, \\ \dot{h} &= \alpha_h(V)(1 - h) - \beta_h(V)h, \\ \dot{n} &= \alpha_n(V)(1 - n) - \beta_n(V)n,\end{aligned}\tag{3.4}$$

kde máme funkce

$$\begin{aligned}\alpha_m(V) &= 0.1 \cdot \frac{25 - V}{e^{\frac{25-V}{10}} - 1}, & \beta_m(V) &= 4 \cdot e^{-\frac{V}{18}}, \\ \alpha_h(V) &= 0.07 \cdot e^{-\frac{V}{20}}, & \beta_h(V) &= \frac{1}{e^{\frac{30-V}{10}} - 1}, \\ \alpha_n(V) &= 0.01 \cdot \frac{10 - V}{e^{\frac{10-V}{10}} - 1}, & \beta_n(V) &= 0.125 \cdot e^{-\frac{V}{80}},\end{aligned}$$

přičemž parametry a proměnné mají následující význam [55, 56, 49]:

V napětí na buněčné membráně (tzv. *lipidové dvouvrstvě*),

C_m elektrická kapacita buněčné membrány,

g_K, g_{Na} elektrická vodivost sodíkových a draslíkových kanálků řízených napětím,

V_K, V_{Na} reverzní potenciály kanálků řízených napětím – potenciály na buněčné membráně, při kterých se otočí proud příslušných iontů v daném kanálku,

g_L, V_L elektrická vodivost, resp. reverzní potenciál, stále otevřených iontových kanálků,

m, h, n pravděpodobnosti spojené s otevřením, resp. zavřením, příslušných kanálků – přesněji řečeno jsou tyto pravděpodobnosti spojeny s aktivací, resp. deaktivací, podjednotek v daných kanálcích [55],

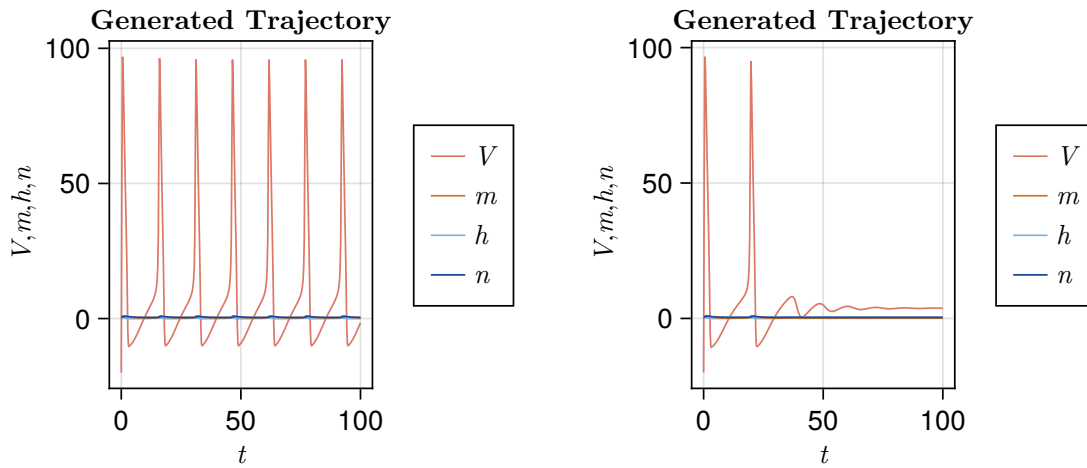
I_e excitační (budící) proud.

Numerické konstanty ve funkcích $\alpha_m, \dots, \alpha_n, \beta_m, \dots, \beta_n$ byly nalezeny experimentálně [48]. Hodnoty parametrů budeme typicky volit podle [50], a to následovně

$$C_m = 1, \quad I_e = \begin{cases} 9 & \text{pro netlumeně oscilující trajektorii} \\ 6 & \text{pro tlumeně oscilující trajektorii} \end{cases} \quad (3.5)$$

$$\begin{aligned} g_{Na} &= 120, & V_{Na} &= 115, \\ g_K &= 36, & V_K &= -12, \\ g_L &= 0.3, & V_L &= 10.6, \end{aligned} \quad (3.6)$$

přičemž jako počáteční podmínku nejčastěji používáme $[-20, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}]$ na časovém intervalu $(0, 100)$. Řešíme-li ODR systém (3.4) pomocí numerické metody Rosenbrock23, dostaneme trajektorie znázorněné na obrázku 3.17.



(a) Netlumeně oscilující trajektorie HH modelu s $I_e = 9$

(b) Tlumeně oscilující trajektorie HH modelu s $I_e = 6$

Obrázek 3.17: Přehled vygenerovaných trajektorií HH modelu ([zdrojový kód](#))

Vzpomeňme si, že FHN model je odůvodněn lineární transformací stavových proměnných HH modelu. V článku [51] navrhuje Richard FitzHugh následující transformaci (u , resp. w , odpovídá proměnné V , resp. W , z FHN modelu)

$$u = V + 36m, \quad w = \frac{n - h}{2}, \quad (3.7)$$

přičemž dále z podobnosti fázových portrétů FHN modelu a takto transformovaného HH modelu tvrdí, že tyto 2 modely patří do „stejně obecné třídy systémů s excitovatelnými oscilacemi“.

Pokusíme se tedy dále v této podkapitole ověřit, zda FitzHughem uvedená lineární transformace opravdu transformuje dynamiku HH modelu na dynamiku FHN modelu a zda není vhodnější některá jiná, nelineární, transformace. Vzhle-

dem k velmi zjednodušenému polynomiálnímu tvaru FHN modelu se dá předpokládat, že úloha nalezení vhodných parametrů FHN modelu z trajektorií nepolynomiálního HH modelu nebude dávat tak dobré výsledky, jako jsme viděli v předchozích dvou studiích 3.1 a 3.2, kdy jsme odvozovali polynomiální tvar z dynamiky modelů taktéž v polynomiálním tvaru.

Transformaci na proměnnou w můžeme odvodit z matice $\mathbf{X}_{h,n} \in \mathbb{R}^{k \times 2}$ obsahující h, n složky simulované trajektorie (uspořádané do sloupců) modelu, jehož transformaci hledáme, tj. v našem případě HH modelu. Pokusíme se nyní transformaci na w odvodit pomocí SVD rozkladu centrované matice $B = \mathbf{X}_{h,n} - \bar{\mathbf{X}}_{h,n}$, kde $\bar{\mathbf{X}}_{h,n}$ je řádkový průměr (tj. průměr přes všechny řádky) definovaný jako [1]

$$\bar{x}_j = \frac{1}{k} \sum_{i=1}^k (\mathbf{X}_{h,n})_{ij}, \quad \bar{\mathbf{X}}_{h,n} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \bar{\mathbf{x}}.$$

Ve výše zobrazeném případě trajektorie s netlumenými oscilacemi, viz 3.17, má SVD rozklad centrované matice B tvar

$$B \approx \underbrace{U \cdot \begin{pmatrix} 3.68 & 0 \\ 0 & 0.30 \end{pmatrix}}_S \cdot \underbrace{\begin{pmatrix} -0.7310 & 0.6824 \\ 0.6824 & 0.7310 \end{pmatrix}}_{V^T},$$

kde V_1 , odpovídá souřadnicové transformaci, která vystihuje nejvíce variability v datech z proměnných h, n (body $[h, n]$ mají největší rozptyl podél této osové přímky – v našem případě pokrývá cca 93 % rozptylu), naopak V_2 , vystihuje pouze odchylky dat od této přímky, což vysvětluje cca 7 % jejich variability.

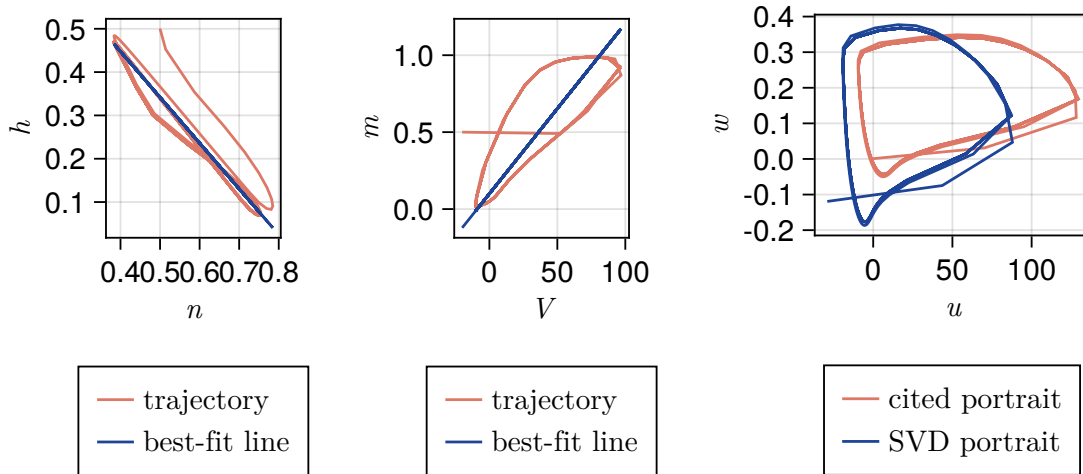
Velmi podobný výsledek dostaneme, pokud body $[h, n]$ proložíme regresní přímkou (ta je vykreslená na obrázku 3.18a). Celkem pro námi zvolené hodnoty parametrů dostaneme, že souřadnicová transformace $w = -0.7310 \cdot h + 0.6824 \cdot n$ dobře aproximuje stavové proměnné h, n , viz obrázek 3.18.

To stejné bychom mohli provést i pro dvojici V, m , u které zjistíme, že kvůli rozdílu velikosti rozptylů v proměnné V a proměnné m , není schopen SVD rozklad „sloučit“ tyto proměnné pomocí lineární transformace. Místo toho dostáváme, že souřadnicová transformace $u = 0.9999 \cdot V + 0.0111 \cdot m$ vystihuje 99 % rozptylu, viz 3.18b. Proto se v článku [51] uvažuje lineární souřadnicová transformace podél křivky separatrix sedlového bodu³.

Pokud se pečlivěji zaměříme na tyto výsledky (viz 3.18), zjistíme, že se proměnná u transformovaného HH modelu pohybuje v intervalu $(-20, 150)$, zatímco proměnná V FHN modelu nabývá hodnot z intervalu $(-4, 4)$, což je patrné např. z obrázku 3.1 – proto bude pravděpodobně obtížné nalézt fyziologické hodnoty parametrů pro FHN model. Stejná situace nastává i pro transformaci získanou pomocí SVD rozkladů trajektorie v proměnných V, m , respektive h, n .

Problém s rozsahem proměnné u , resp. v menší míře i proměnné w , lze jistě vyřešit přeškálováním použitých transformací. Potom bychom dostali transformovanou trajektorii, která se pohybuje na *podobném* atraktoru – to lze vidět i na obrázku

³Informace o sedlovém bodu lze najít [zde](#) a o separatrix [zde](#).



(a) Lineární transformace proměnných h, n na w (b) Lineární transformace proměnných V, m na u (c) Znáznornění transformované trajektorie (podle FitzHugha i pomocí SVD rozkladů)

Obrázek 3.18: Přehled FitzHughovy lineární transformace HH modelu ([zdrojový kód](#))

3.18c (přeškálování os nezmění „tvar“ trajektorie v prostoru u, w). Nicméně, co nelze vidět, je závislost u, w na čase – pokud bychom si vykreslili $u(t), w(t)$ transformovaného HH modelu a $V(t), W(t)$ FHN modelu (s vhodnými parametry), tak zjistíme, že FHN model má tendenci oscilovat rychleji než transformovaný HH model. To stejné je patrné i na obrázcích 3.19 a 3.21, znázorňující řešení relaxované varianty tohoto problému, kterou uvádíme níže.

Navíc si můžeme uvědomit, že rozhodnutí hledat u , resp. w , ve tvaru $u = \alpha V + \beta m$, resp. $w = \gamma h + \delta n$, je při hledání dobré aproximace FHN modelu zbytečně omezující. Doposud jsme se nezabývali ani otázkou, co vlastně myslíme pod „dobrou aproximací FHN modelu“.

Uvažujme nyní trajektorii HH modelu s netlumenými oscilacemi (3.5). Nechť $k \in \mathbb{N}$ je počet měření, $\mathbf{H} \in \mathbb{R}^{k \times 4}$ matice trajektorie HH modelu⁴ (měření jednotlivých proměnných jsou uspořádány do sloupců), $\mathbf{f}$ funkce zadávající pravou stranu FHN modelu (1.2), $\mathbf{u} = (V, W)$ stavové proměnné FHN modelu, $\mathbf{p} = (i_e, a, b, c, d)$ vektor jeho parametrů tak, že platí

$$\dot{\mathbf{u}} = \mathbf{f}(\mathbf{u}; \mathbf{p}).$$

Dále označme $\mathbf{u}(t; \mathbf{p}, \mathbf{u}_0, t_0)$ stav systému FHN modelu v čase $t \in \mathbb{R}$ s počáteční podmínkou $\mathbf{u}_0 \in \mathbb{R}^2$ a počátečním časem $t_0 \in \mathbb{R}$. Jelikož je systém příslušný FHN modelu autonomní, bez újmy na obecnosti můžeme uvažovat $t_0 = 0$ (v opačném případě zavedeme substituci $\tau = t - t_0$) [9]. Proto použijme jedno-

⁴V předchozí části této sekce jsme ji kvůli obecnosti značili $\mathbf{X}$.

dušší značení $\mathbf{u}(t; \mathbf{p}, \mathbf{u}_0)$. Předpokládejme, že měření trajektorie $\mathbf{H}$ proběhly v časech $T = \{t_1, \dots, t_k\}$, a označme

$$\mathbf{u}(T; \mathbf{p}, \mathbf{u}_0) = \begin{pmatrix} \mathbf{u}(t_1; \mathbf{p}, \mathbf{u}_0) \\ \vdots \\ \mathbf{u}(t_k; \mathbf{p}, \mathbf{u}_0) \end{pmatrix}$$

trajektorii FHN modelu zaznamenanou v těchto časech $t_i \in T$ pro $i = 1, \dots, k$.

V neposlední řadě zavedme matici transformace $\mathbf{\Lambda} \in \mathbb{R}^{4 \times 2}$ HH modelu na FHN model. Tím pádem nalezení optimální transformace Hodgkinova–Huxleyho modelu na FitzHughův–Nagumův model s vhodnými parametry a počáteční podmínkou můžeme matematicky formulovat jako optimalizační problém

$$\min_{\mathbf{\Lambda}, \mathbf{p}, \mathbf{u}_0} (\text{loss}(\mathbf{u}(T; \mathbf{p}, \mathbf{u}_0), \mathbf{H}\mathbf{\Lambda}) + \mu R(\mathbf{p})), \quad (3.8)$$

kde $R(\mathbf{p})$ je regularizační funkce parametru $\mathbf{p}$ (nebude-li řečeno jinak, uvažujeme $R(\cdot) = \|\cdot\|_2$), μ je parametr regularizace⁵ a $\text{loss}(\mathbf{x}, \mathbf{y})$ je námi definovaná ztrátová funkce. Experimentálně jsme určili vhodný tvar ztrátové funkce na

$$\text{loss}(\mathbf{x}, \mathbf{y}) = \frac{1}{k} \|\mathbf{x} - \mathbf{y}\|_2^2 + \alpha \left| 1 - \frac{\sigma(\mathbf{x})}{\sigma(\mathbf{y})} \right| + \beta \left| 1 - \frac{\sigma(\mathbf{x}_a)}{\sigma(\mathbf{x}_b)} \right|, \quad (3.9)$$

přičemž $\mathbf{x} \in \mathbb{R}^{k \times 2}$ zastupuje simulovanou trajektorii FHN modelu pro dané parametry a počáteční podmínku a $\mathbf{y} \in \mathbb{R}^{k \times 2}$ transformovanou trajektorii HH modelu. Parametry α, β jsou v rámci optimalizace pevné (volíme je před započítáním optimalizace). Význam symbolů a členů v rovnici (3.9) je následující:

$\sigma(\mathbf{z})$ směrodatná odchylka dat $\mathbf{z}$ definovaná jako

$$\sigma(\mathbf{z}) = \sqrt{\frac{1}{k-1} \sum_{i=1}^k \|\mathbf{z}_i - \bar{\mathbf{z}}\|_2^2},$$

kde $\bar{\mathbf{z}} \in \mathbb{R}^2$ značí průměr dat $\mathbf{z}$,

$\mathbf{x}_a$ první polovina dat $\mathbf{x}$ – pro $\mathbf{x} \in \mathbb{R}^{k \times 2}$ definujeme jako

$$\mathbf{x}_a = \begin{pmatrix} \mathbf{x}_{1,\cdot} \\ \vdots \\ \mathbf{x}_{l,\cdot} \end{pmatrix}, \quad \text{kde } l = \left\lfloor \frac{k}{2} \right\rfloor = \begin{cases} \frac{k}{2} & \text{pro } k \text{ sudé,} \\ \frac{k-1}{2} & \text{pro } k \text{ liché,} \end{cases}$$

$\mathbf{x}_b$ druhá polovina $\mathbf{x}$, definovaná obdobně jako $\mathbf{x}_a$,

$\frac{1}{k} \|\mathbf{x} - \mathbf{y}\|_2^2$ průměrná vzdálenost simulované a transformované trajektorie – normu

⁵Penalizováním complexity či velikosti parametru $\mathbf{p}$ zajišťujeme určitou numerickou stabilitu při řešení FHN modelu. Parametr regularizace μ je v rámci optimalizace pevný – volíme jej před jejím započítáním.

$\|\mathbf{x} - \mathbf{y}\|_2^2$ průměrujeme, abychom ošetřili situaci, kdy se nám nepodaří simulovat trajektorii FHN modelu pro celé T . V takovém případě používáme zkrácené T' dlouhé tak, jak nám simulace FHN modelu dovolí. Jinak bychom dostali kratší trajektorii, která by zkreslovala skutečnou „vhodnost“ nalezené transformace $\mathbf{\Lambda}$, parametrů $\mathbf{p}$ a počáteční podmínky $\mathbf{u}_0$,

$\left|1 - \frac{\sigma(\mathbf{x})}{\sigma(\mathbf{y})}\right|$ odpovídá poměru směrodatných odchylek simulované a transformované trajektorie a vynucuje rovnost jejich rozptylů (resp. směrodatných odchylek)

$\left|1 - \frac{\sigma(\mathbf{x}_a)}{\sigma(\mathbf{x}_b)}\right|$ odpovídá poměru směrodatných odchylek první a druhé poloviny simulované trajektorie $\mathbf{x}$ – v případě netlumených oscilací trajektorie HH modelu tímto vynucujeme obdobné oscilace i v FHN modelu⁶,

α, β parametry vážící rovnost rozptylů trajektorií $\mathbf{x}$ a $\mathbf{y}$, resp. konstantnost rozptylu trajektorie $\mathbf{x}$.

Uvědomme si, že optimalizační problém (3.8) jistě nemá konvexní účelovou funkci, a tedy na jeho řešení nemůžeme použít ADMM algoritmus představený v sekci 1.3.2. Proto jej budeme řešit **Nelderovou–Meadovou** metodou [57], která nevyžaduje znalost gradientu (který by v tomto případě bylo velmi obtížné spočítat). Pro implementaci použijeme **julia** knihovnu **Optim.jl** [58].

V tuto chvíli nám již stačí zformulovat postup, jak se budeme pokoušet vhodnou transformací a parametry FHN modelu nalézt. V první řadě si definujeme počáteční hodnoty pro optimalizaci (jako výchozí používáme FitzHughovu transformaci)

$$\mathbf{\Lambda}_i = \begin{pmatrix} 1 & 0 \\ 36 & 0 \\ 0 & 0.5 \\ 0 & -0.5 \end{pmatrix}, \quad \mathbf{p}_i = (0.45, 1.0, 0.8, 0.7, 0.8),$$

přičemž jako výchozí počáteční podmínku $\mathbf{u}_{0_i}$ volíme $\mathbf{u}_{0_i} = \mathbf{H}_{1,\cdot} \cdot \mathbf{\Lambda}_i$. Dále je nutné zvolit parametry optimalizace, v našem případě

$$\alpha = 50, \quad \beta = 1, \quad \mu = 0.1, \quad \text{pro netlumené oscilace,} \quad (3.10)$$

$$\alpha = 1, \quad \beta = 0, \quad \mu = 0.1, \quad \text{pro tlumené oscilace,} \quad (3.11)$$

které jsme určili experimentálně ze simulací. Připomeňme, že $\mathbf{H}$ označuje trajektorii HH modelu vykazující netlumené oscilace (3.5). S takto určenými hodnotami nalezneme po nejvýše 10 000 iteracích Nelderovy–Meadovy metody řešení

$$\mathbf{\Lambda} = \begin{pmatrix} 0.065 & 0.038 \\ -5.725 & 0.002 \\ 0.028 & 0.550 \\ 0.036 & -0.700 \end{pmatrix}, \quad \mathbf{u}_0 = [-1.407, 0.024], \quad (3.12)$$

$$\mathbf{p} = \underbrace{(0.948, 0.820, 0.586, 0.485, 0.609)}_{(i_e, a, b, c, d)},$$

⁶V opačném případě má optimalizace tendenci nalézt *skoro-konstantní* trajektorii na hladině průměru.

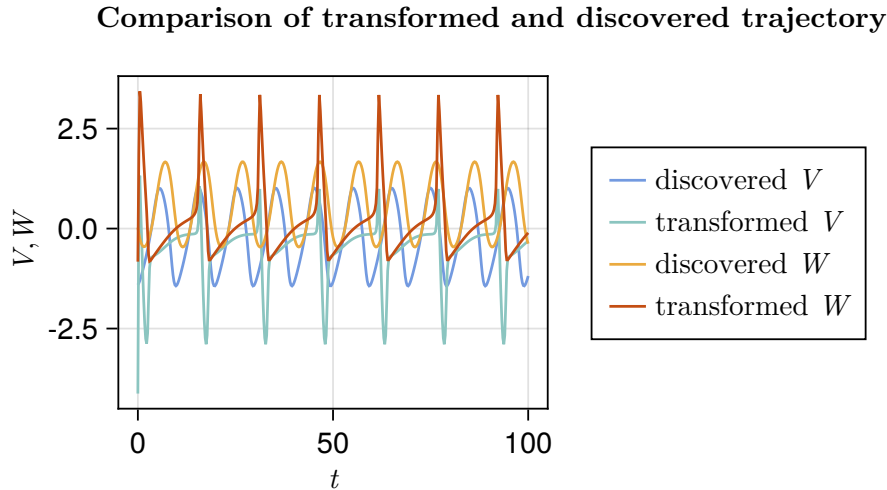
přičemž trajektorie $\mathbf{u}(T; \mathbf{p}, \mathbf{u}_0)$ vystihuje $\mathbf{H}\mathbf{\Lambda}$ s koeficientem determinace $R^2 = -0.852$. Srovnání simulované a transformované trajektorie můžeme vidět na obrázcích 3.19 a 3.20a. Kvalitu nalezené transformace $\mathbf{\Lambda}$ dále ověříme pomocí trajektorie HH modelu s tlumenými oscilacemi (3.6), kterou budeme značit $\mathbf{H}'$ – potřebujeme k ní avšak najít vhodné parametry $\mathbf{p}_f$ FHN modelu a jeho počáteční podmínku $\mathbf{u}_{0f}$, což bude řešení zjednodušené varianty problému (3.8)

$$\min_{\mathbf{p}_f, \mathbf{u}_{0f}} \left(\text{loss} \left(\mathbf{u}(T; \mathbf{p}_f, \mathbf{u}_{0f}), \mathbf{H}'\mathbf{\Lambda} \right) + \mu R(\mathbf{p}_f) \right), \quad (3.13)$$

pro $\mathbf{\Lambda}$ pevné již nalezené, přičemž navíc pokládáme parametry $\alpha = 1, \beta = 0$ podle (3.11)⁷. Problém (3.13), na který opět aplikujeme Nelderovu–Meadovu metodu, má řešení

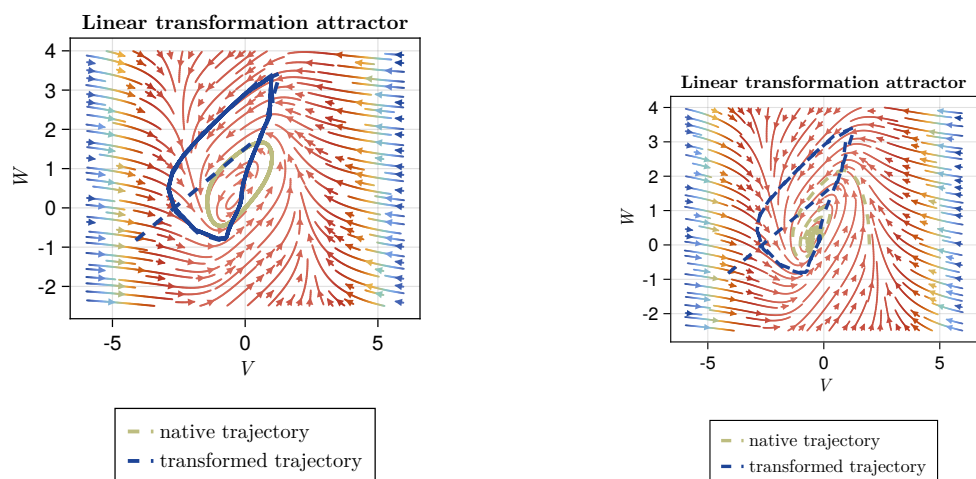
$$\mathbf{p}_f = (1.040, 1.090, 0.771, 0.727, 0.691), \quad \mathbf{u}_{0f} = [1.966, 0.016]. \quad (3.14)$$

Koeficient determinace R^2 je v tomto případě roven $R^2 = -0.203$ a porovnání těchto trajektorií lze vidět obrázcích 3.20b a 3.21.



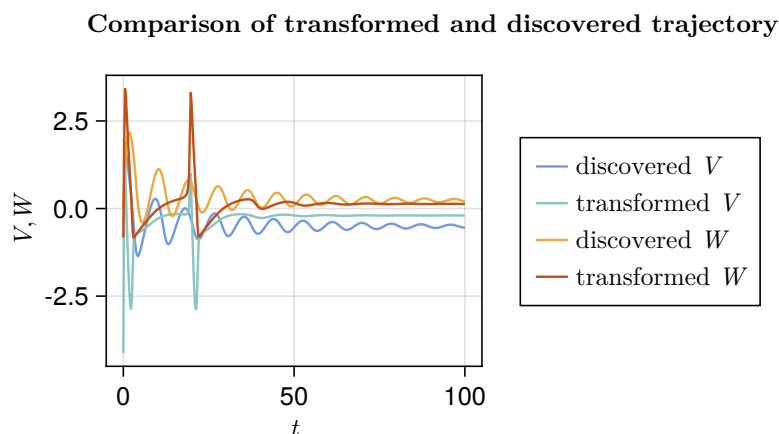
Obrázek 3.19: Srovnání časového průběhu simulované trajektorie FHN modelu a transformovaného HH modelu pro netlumené oscilace ([zdrojový kód](#))

⁷Původní trajektorie HH modelu s tlumenými oscilacemi nemá konstantní rozptyl, tedy to nebudeme vyžadovat ani po simulované trajektorii.



(a) Fázový portrét nalezeného FHN modelu pro HH model s netlumenými oscilacemi (b) Fázový portrét nalezeného FHN modelu pro HH model s tlumenými oscilacemi

Obrázek 3.20: Trajektorie zobrazené ve fázovém portréту nalezeného FHN modelu ([zdrojový kód](#))



Obrázek 3.21: Srovnání časového průběhu simulované trajektorie FHN modelu a transformovaného HH modelu pro tlumené oscilace ([zdrojový kód](#))

Na obrázcích výše je patrná zmiňovaná tendence trajektorie FHN modelu oscilovat rychleji, než je tomu u transformovaného HH modelu.

Ačkoliv jsme pomocí výše nalezených lineárních transformací zobrazili HH model „blízko“ vhodného FHN modelu, stále bychom si mohli přát, aby si tyto modely odpovídaly mírně lépe, či aby oba modely oscilovaly přibližně stejnou rychlostí.

Použijme proto obdobnou myšlenku jako u SINDy metody, a to vytvoření knihovny kandidátních funkcí Θ , pomocí níž transformuje $\mathbf{H}$, resp. $\mathbf{H}'$, tak, aby bylo jednodušší nalézt dobrou transformační matici $\mathbf{A}$. Tuto úlohu můžeme mate-

maticky zapsat jako zobecnění optimalizačního problému (3.8)

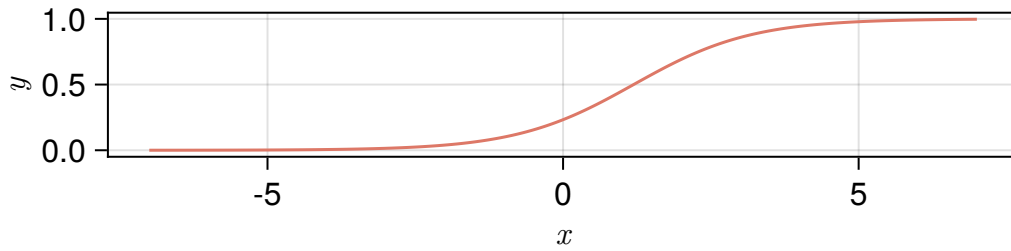
$$\min_{\Lambda, \mathbf{p}, \mathbf{u}_0, \mathbf{q}} (\text{loss}(\mathbf{u}(T; \mathbf{p}, \mathbf{u}_0), \Theta(\mathbf{H}; \mathbf{q})\Lambda) + \mu R(\mathbf{p})), \quad (3.15)$$

kde význam funkcí loss , R zůstává stejný a Θ značí knihovnu kandidátních funkcí, která závisí na vektoru parametrů $\mathbf{q}$ – tento vektor je také součástí optimalizace. První si ukažme příklad nelineární transformace s přidáním *inverzními sigmoidami* pro stavové proměnné m, h, n – ty totiž reprezentují pravděpodobnosti a tedy nabývají hodnot $[0, 1]$, ale obě proměnné FHN modelu mohou (teoreticky) nabývat libovolné hodnoty z $\mathbb{R}$.

Nechť $S : \mathbb{R} \rightarrow (0, 1)$ je funkce tvaru

$$S(x; q, r) = \frac{1}{1 + qe^{-(x-r)}},$$

kde $q_1 \neq 0$, pak pro potřeby této práce tuto funkci nazveme *sigmoida*⁸, viz obrázek 3.22.



Obrázek 3.22: Příklad sigmoidy $S(x; 2, \frac{1}{2})$ na intervalu $[-7, 7]$ ([zdrojový kód](#))

Všimněme si, že sigmoida je bijekcí mezi $\mathbb{R}$ a intervalem $(0, 1)$. Tím pádem bude její inverze zobrazení $\mathbb{R} \rightarrow (0, 1)$, které můžeme použít na transformaci pravděpodobností m, h, n . Proto počítejme

$$\begin{aligned} y &= S(x; q, r) = \frac{1}{1 + qe^{-(x-r)}} \\ \frac{1}{y} &= 1 + qe^{-(x-r)} \\ \frac{1-y}{yq} &= e^{-(x-r)} \\ -\ln\left(\frac{1-y}{yq}\right) &= x-r \\ x &= \ln\left(\frac{yq}{1-y}\right) + r = S^{-1}(y; q, r), \end{aligned}$$

přičemž funkci S^{-1} budeme nazývat *inverzní sigmoida*.

⁸Sigmoidu lze definovat obecněji, více [zde](#).

Zkonstruuje nyní knihovnu kandidátních funkcí $\Theta_s(\mathbf{H}; \mathbf{q}, \mathbf{r})$ pro $\mathbf{q}, \mathbf{r} \in \mathbb{R}^3$ jako

$$\Theta_s(\mathbf{H}; \mathbf{q}, \mathbf{r}) = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots \\ \mathbf{H}_{i,2} & S^{-1}(m; \mathbf{q}_1, \mathbf{r}_1) & S^{-1}(h; \mathbf{q}_2, \mathbf{r}_2) & S^{-1}(n; \mathbf{q}_3, \mathbf{r}_3) \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}, \quad (3.16)$$

kde m odpovídá $\mathbf{H}_{i,2}$ pro $i = 1, \dots, k$ a obdobně pro zbylé proměnné. Poté můžeme zapsat optimalizační problém s touto knihovnou kandidátů (3.16) jako

$$\min_{\Lambda_s, \mathbf{p}_s, \mathbf{u}_{0_s}, \mathbf{q}_s, \mathbf{r}_s} (\text{loss}(\mathbf{u}(T; \mathbf{p}_s, \mathbf{u}_{0_s}), \Theta_s(\mathbf{H}; \mathbf{q}_s, \mathbf{r}_s) \Lambda_s) + \mu R(\mathbf{p}_s)), \quad (3.17)$$

přičemž $\Lambda_s \in \mathbb{R}^{7 \times 2}$. Trajektorii Hodgkinova–Huxleyho modelu, ke které přidáme inverzní sigmoidy proměnných m, h, n a poté ji transformuje pomocí Λ_s , budeme zkráceně označovat jako σ -transformovanou. Minimalizační úloha (3.17) s parametry (3.10) je opět řešitelná Nelderovou–Meadovou metodou a jejím výsledkem je v případě netlumených oscilací (3.5)

$$\Lambda_s = \begin{pmatrix} 0.094 & 0.004 \\ -9.049 & 0.012 \\ -0.003 & 0.725 \\ 0.008 & -0.559 \\ 0.214 & 0.317 \\ 0.980 & 0.838 \\ 1.415 & 1.049 \end{pmatrix}, \quad \mathbf{u}_{0_s} = [-1.774, 0.017], \quad (3.18)$$

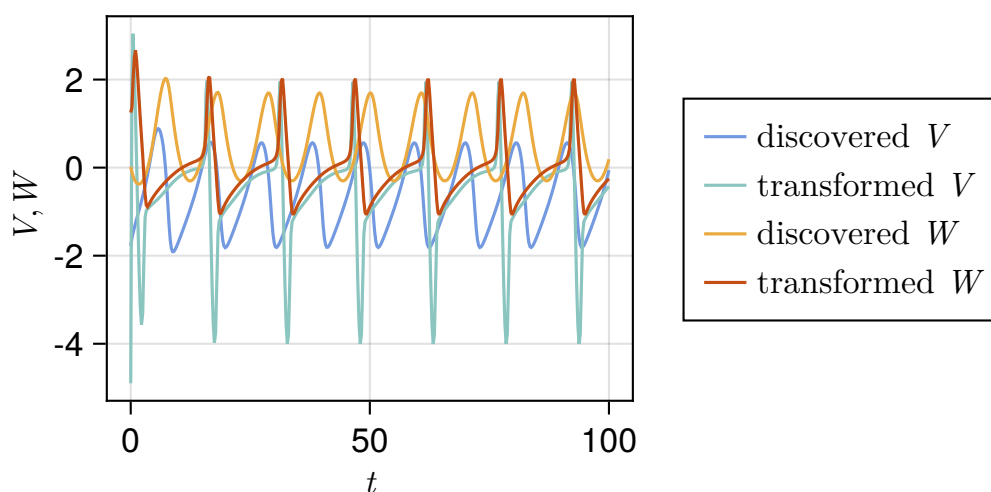
$$\mathbf{p}_s = (0.508, 1.125, 0.422, 0.946, 0.689),$$

$$\mathbf{q}_s = (1.483, 1.672, 1.889), \quad \mathbf{r}_s = (0.021, 0.002, 0.004).$$

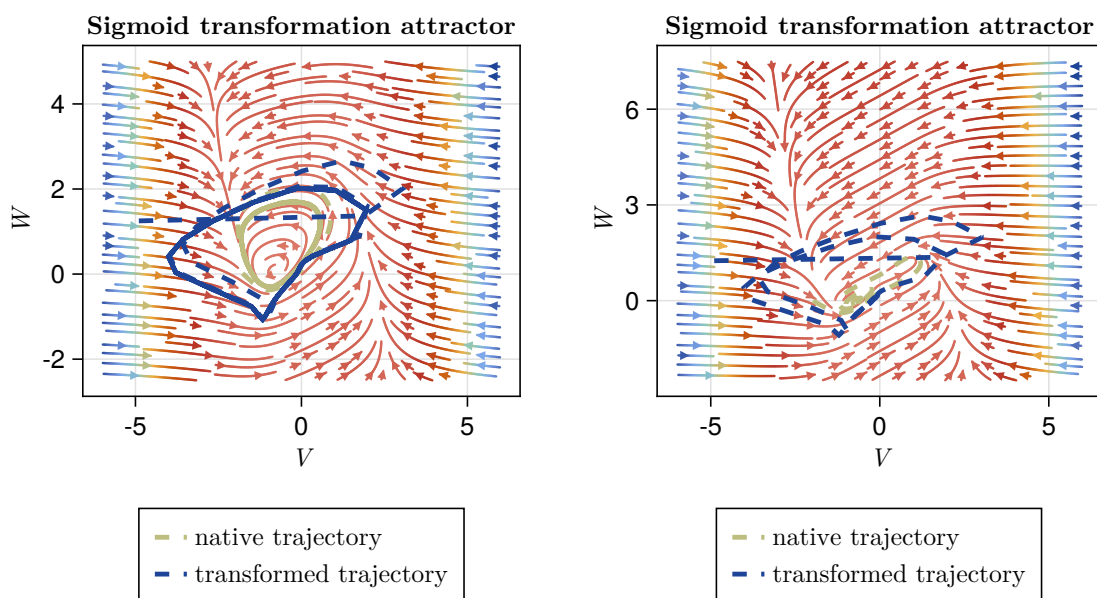
Trajektorie nalezeného FHN modelu s parametry $\mathbf{p}_s$ a počáteční podmínkou $\mathbf{u}_{0_s}$, viz (3.18), má vůči σ -transformované trajektorii $\mathbf{H}$ koeficient determinace roven $R^2 = -0.719$. To značí mírně lepší kvalitu nalezeného modelu a příslušné σ -transformace, než pro lineární transformaci a jí odpovídající FHN model (3.12). Srovnání trajektorií nalezeného FHN modelu a σ -transformovaného HH modelu lze nalézt na obrázcích 3.23 a 3.24a.

Obdobným postupem pro trajektorii $\mathbf{H}'$ s tlumenými oscilacemi (3.6) a parametry optimalizace (3.11) dostaneme FHN model odpovídající σ -transformované trajektorii $\mathbf{H}'$ s koeficientem determinace $R^2 = -0.389$. Obě tyto trajektorie jsou znázorněné na obrázcích 3.24b a 3.25.

Comparison of transformed and discovered trajectory



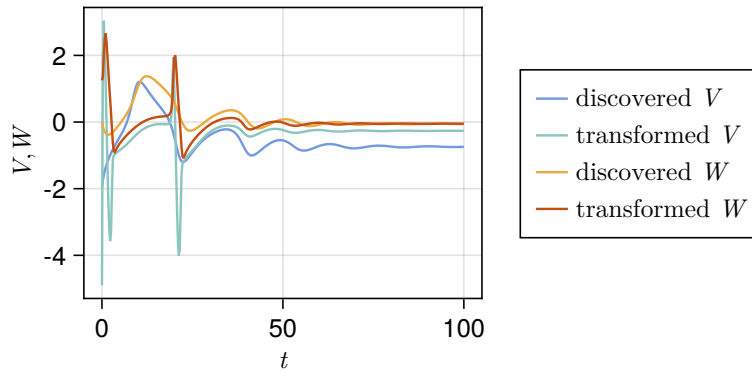
Obrázek 3.23: Srovnání časového průběhu simulované trajektorie FHN modelu a σ -transformovaného HH modelu pro netlumené oscilace ([zdrojový kód](#))



(a) Fázový portrét nalezeného FHN modelu pro HH model s netlumenými oscilacemi (b) Fázový portrét nalezeného FHN modelu pro HH model s tlumenými oscilacemi

Obrázek 3.24: Trajektorie zobrazené ve fázovém portrétu nalezeného FHN modelu ([zdrojový kód](#))

Comparison of transformed and discovered trajectory



Obrázek 3.25: Srovnání časového průběhu simulované trajektorie FHN modelu a σ -transformovaného HH modelu pro tlumené oscilace ([zdrojový kód](#))

Jako poslední vyzkoušíme přidat do knihovny kandidátních prvků Θ_e exponenciálu, tj.⁹

$$\Theta_e(\mathbf{H}; \mathbf{q}) = \begin{pmatrix} \vdots & \vdots \\ \mathbf{H}_{i,\cdot} & e^{\langle \mathbf{H}_{i,\cdot}, \mathbf{q} \rangle} \\ \vdots & \vdots \end{pmatrix}, \quad (3.19)$$

přičemž $\mathbf{H}_{i,\cdot} \in \mathbb{R}^4$ pro $i = 1, \dots, k$ a $\mathbf{q} \in \mathbb{R}^4$ je vektor parametrů knihovny Θ_e . Takto dostáváme optimalizační problém

$$\min_{\Lambda_e, \mathbf{p}_e, \mathbf{u}_{0_e}, \mathbf{q}_e} (\text{loss}(\mathbf{u}(T; \mathbf{p}_e, \mathbf{u}_{0_e}), \Theta_e(\mathbf{H}; \mathbf{q}_e) \Lambda_e) + \mu R(\mathbf{p}_e)), \quad (3.20)$$

kde $\Lambda_e \in \mathbb{R}^{5 \times 2}$. Trajektorii HH modelu, kterou obohatíme o exponenciálu pomocí knihovny Θ_e a pak transformujeme maticí Λ_e , nazveme *e-transformovaná*.

Parametry optimalizace volíme podle (3.10) s výjimkou β , pro které použijeme hodnotu $\beta = 10$, a úlohu (3.20) řešíme opět pomocí Nelderova–Meadova algoritmu, což nám dá výsledek

$$\Lambda_e = \begin{pmatrix} 0.114 & 0.004 \\ -10.509 & 0.013 \\ -0.009 & 0.658 \\ 0.016 & -0.525 \\ 0.703 & 1.074 \end{pmatrix}, \quad (3.21)$$

$$\mathbf{u}_{0_e} = [-5.036, 0.055], \quad \mathbf{q}_e = (-0.022, 0.922, 0.827, 1.140), \\ \mathbf{p}_e = (0.520, 1.697, 0.659, 1.383, 1.345),$$

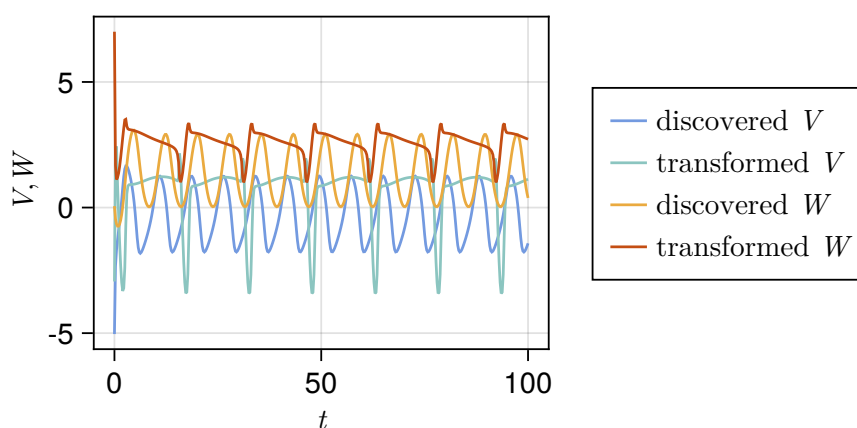
pro něž si trajektorie FHN modelu a *e-transformovaného* HH modelu odpovídají s koeficientem determinace $R^2 = -0.730$. Opět vidíme mírné zlepšení oproti lineární (3.12) transformaci, ale nepatrné zhoršení oproti σ -transformaci (3.18) (a

⁹Skalární součin 2 vektorů $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ označujeme $\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}_1 \mathbf{b}_1 + \dots + \mathbf{a}_n \mathbf{b}_n$.

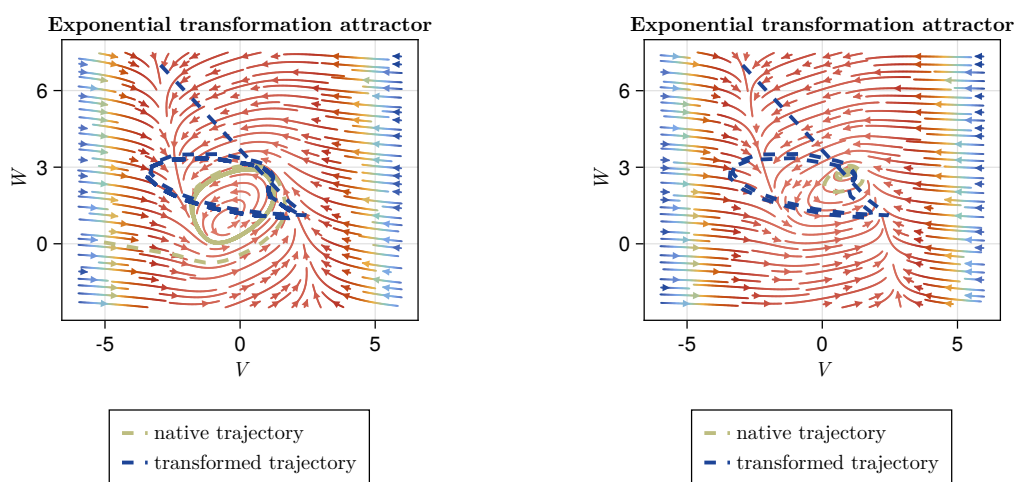
jím příslušným FHN modelům). Srovnání trajektorií nalezeného FHN modelu a e -transformovaného HH modelu lze spatřit na obrázcích 3.26 a 3.27a.

V neposlední řadě provedme stejný proces i pro trajektorii H' s tlumenými oscilacemi (3.6), přičemž optimalizační parametry pro úlohu (3.20) volíme podle (3.11). Výsledná trajektorie vhodného FitzHughova–Nagumova modelu a e -transformovaná trajektorie H' Hodgkinova–Huxleyho modelu si odpovídají s koeficientem determinace $R^2 = 0.660$, přičemž jejich srovnání je ukázané na obrázcích 3.27b a 3.28.

Comparison of transformed and discovered trajectory



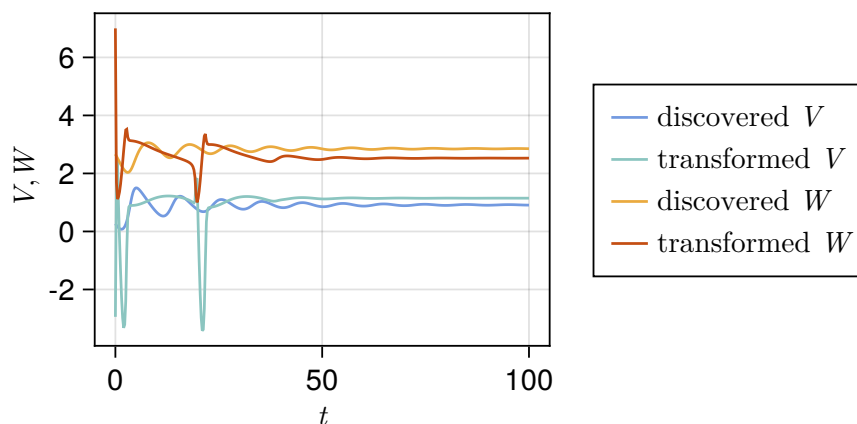
Obrázek 3.26: Srovnání časového průběhu simulované trajektorie FHN modelu a e -transformovaného HH modelu pro netlumené oscilace ([zdrojový kód](#))



(a) Fázový portrét nalezeného FHN modelu pro HH model s netlumenými oscilacemi (b) Fázový portrét nalezeného FHN modelu pro HH model s tlumenými oscilacemi

Obrázek 3.27: Trajektorie zobrazené ve fázovém portrétu nalezeného FHN modelu ([zdrojový kód](#))

Comparison of transformed and discovered trajectory



Obrázek 3.28: Srovnání časového průběhu simulované trajektorie FHN modelu a e -transformovaného HH modelu pro tlumené oscilace ([zdrojový kód](#))

Srovnání všech uvedených transformací a modelů je nelehký úkol, neboť každá varianta nabízí něco jiného. Z lineární transformace dostáváme interpretovatelné zobrazení Hodgkinova–Huxleyho modelu na FitzHughův–Nagumův model. Naopak z obrázku 3.25 je patrné, že od času cca $t = 30$ velmi dobře odpovídá dynamika vhodného FHN modelu a σ -transformace HH modelu. V neposlední řadě e -transformace nejlépe řeší různé rychlosti oscilace těchto dvou modelů, kdy z obrázku 3.26 můžeme vidět, že FHN model osciluje skoro přesně dvakrát rychleji než e -transformovaný HH model.

Závěr

Schopnost odvodit nelineární dynamiku systému z dat z něj pocházejících je velmi silný nástroj, který má mnohá využití nejen v praxi, ale i ve spoustě vědních oborů. V této práci jsem představil *řídkou identifikaci nelineární dynamiky (SINDy)* jako jeden z možných přístupů k tomuto problému.

V první kapitole jsem vybudoval nutný teoretický základ potřebný pro SINDy metodu. Ten jsem v druhé kapitole použil na matematickou formulaci SINDy metody a poté jsem předvedl jednoduchou a názornou implementaci v jazyce **julia** s různými optimalizačními metodami a volbami knihovny kandidátních funkcí. Konec této kapitoly jsem věnoval problémům, na které lze při použití SINDy metody narazit. V poslední kapitole jsem uvedl 2 simulační studie na vybraných modelech. FitzHughův–Nagumův model jsem zvolil kvůli oscilacím, které v něm vznikají, a Lorenzův model kvůli jeho chaotickému chování. V poslední sekci se věnuji transformaci dynamiky Hodgkinova–Huxleyho modelu na jednodušší FitzHughův–Nagumův model.

Největší přínos této práce vidím v originální modifikaci *DSTLS* již existující metody *STLS*. Tuto novou metodu jsem srovnal s ostatními optimalizačními metodami, prozkoumal jsem její chování na datech ovlivněných šumem a poté jsem ji aplikoval na výše zmíněné vybrané modely. Metodu *DSTLS* spolu s dalšími výsledky a analýzami plánuji publikovat ve vědeckém časopise.

Za neméně podstatnou považuji poslední sekci, ve které objasňuji FitzHughovu lineární transformaci Hodgkinova–Huxleyho modelu, dále pak představuji obecnější metodu získání vhodné lineární i nelineární transformace a ilustruji ji pro různé transformace s tlumenou i netlumenou oscilací v použité trajektorii Hodgkinova–Huxleyho modelu.

Hodnotu také spatřuji v mé implementaci SINDy v jazyce **julia**, neboť jsem toho názoru, že napsaný kód je čitelný, jednoduchý a dobře pochopitelný, ale zároveň zahrnuje celý proces hledání modelu.

Již zmiňované porovnání optimalizačních metod v závislosti na velikosti šumu je taktéž originálním výsledkem. Domnívám se, že tuto analýzu lze využít při rozhodování, kterou optimalizační metodu použít, máme-li výrazně zašuměná data. V neposlední řadě tato práce přináší ucelené a vcelku podrobné pojetí tématu dynamických systémů řízených daty jak s vysvětlením potřebné teorie, tak i různorodými aplikačními ukázkami.

Přílohy

```
struct CustomLibrary{T<:AbstractArray} <: Library
    functions::T
end

Base.length(library::CustomLibrary) = length(library.functions)
Base.iterate(library::CustomLibrary; kwargs...) =
    ↪ Base.iterate(library, 0; kwargs...)

function Base.iterate(library::CustomLibrary, index; kwargs...)
    index += 1
    if index > length(library.functions)
        return nothing
    end

    func = X -> eachstate(system_state ->
    ↪ library.functions[index](system_state), X; kwargs...)
    return (func, index)
end
```

Algoritmus 10: Definice vlastní knihovny kandidátních funkcí

```

struct MonomialLibrary{T<:Integer} <: PredefinedLibrary
    degree::T
    variables::T
end

Base.length(library::MonomialLibrary) = library.degree *
↳ library.variables + 1

function Base.iterate(library::MonomialLibrary; stateful=false,
↳ kwargs...)
    state = zeros(library.variables)
    func = X -> eachstate(system_state -> prod(system_state .^
↳ state), X; kwargs...)
    return stateful ? (state, (0, library.variables, stateful))
↳ : (func, (0, library.variables, stateful))
end

function Base.iterate(library::MonomialLibrary, state;
↳ kwargs...)
    power, var, stateful = state
    power += 1
    if power > library.degree
        var -= 1
        power = 1
    end
    if var == 0
        return nothing
    end

    func = X -> eachstate(system_state ->
↳ system_state[var]^power, X; kwargs...)
    if !stateful
        return (func, (power, var, stateful))
    end

    power_vector = zeros(library.variables)
    power_vector[var] = power
    return (power_vector, (power, var, stateful))
end

```

Algoritmus 11: Definice knihovny monomiálních funkcí

```

struct PolynomialLibrary{T<:Integer} <: PredefinedLibrary
    degree::T
    variables::T
end

is_malformed(state::Vector, library::PolynomialLibrary) =
    ↳ sum(state) > library.degree || any(state .> library.degree)

Base.length(library::PolynomialLibrary) = sum(binomial(k +
    ↳ library.variables - 1, library.variables - 1) for k in
    ↳ 0:(library.degree))
function Base.iterate(library::PolynomialLibrary;
    ↳ stateful=false, kwargs...)
    state = zeros(library.variables)
    func = X -> eachstate(system_state -> prod(system_state .^
    ↳ state), X; kwargs...)
    return stateful ? (state, (state, stateful)) : (func,
    ↳ (state, stateful))
end

function Base.iterate(library::PolynomialLibrary, state;
    ↳ kwargs...)
    state, stateful = state
    index = length(state)
    state[index] += 1
    while is_malformed(state, library)
        state[index] = 0
        index -= 1
        if index == 0
            return nothing
        end
        state[index] += 1
    end

    func = X -> eachstate(system_state -> prod(system_state .^
    ↳ state), X; kwargs...)
    return stateful ? (state, (state, stateful)) : (func,
    ↳ (state, stateful))
end

```

Algoritmus 12: Definice knihovny polynomiálních funkcí

Seznam použité literatury

1. BRUNTON, Steven L.; KUTZ, J. Nathan. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. Dostupné z doi: [10.1017/9781108380690](https://doi.org/10.1017/9781108380690).
2. BEZANSON, Jeff; EDELMAN, Alan; KARPINSKI, Stefan; SHAH, Viral B. **julia**: A fresh approach to numerical computing. *SIAM review*. 2017, roč. 59, č. 1, s. 65–98. Dostupné také z: <https://doi.org/10.1137/141000671>.
3. LUK, Adam; MÁCHA, Vašek; MAŠKOVÁ, Michaela. **julia** pro optimalizaci a učení. 2021. Dostupné z eprint: <https://julioteachingctu.github.io/Julia-for-Optimization-and-Learning/stable/>.
4. JOHNSON, Steven G. *Efficient way of doing linear regression*. 2020. Dostupné také z: <https://discourse.julialang.org/t/efficient-way-of-doing-linear-regression/31232/33>.
5. JOHNSON, Steven G. *Inverse and backslash \ division*. 2020. Dostupné také z: <https://discourse.julialang.org/t/inverse-and-backslash-division/44680/4>.
6. CHRIST, Simon; SCHWABENEDER, Daniel; RACKAUCKAS, Christopher; BORREGAARD, Michael Krabbe; BRELOFF, Thomas. *Plots.jl – a user extendable plotting API for the julia programming language*. arXiv, 2022. Dostupné z doi: [10.48550/ARXIV.2204.08775](https://doi.org/10.48550/ARXIV.2204.08775).
7. DANISCH, Simon; KRUMBIEGEL, Julius. Makie.jl: Flexible high-performance data visualization for **julia**. *Journal of Open Source Software*. 2021, roč. 6, č. 65, s. 3349. Dostupné z doi: [10.21105/joss.03349](https://doi.org/10.21105/joss.03349).
8. PŘIBYLOVÁ, Lenka. *Nelineární dynamika a její aplikace*. 1. vyd. Brno: Masarykova univerzita, 2012. ISBN 978-80-210-5969-6. Dostupné také z: <https://is.muni.cz/auth/el/sci/jaro2022/M6201/um/NelinDyn.pdf>.
9. POSPÍŠIL, Zdeněk; PŘIBYLOVÁ, Lenka. *Matematická biologie: e-learningová učebnice [online]*. Spojité deterministické modely I [elektronická verze "online"]. Brno: Masarykova univerzita, 2015. ISBN 978-80-210-8095-9. Dostupné také z: <http://portal.matematickabiologie.cz/index.php?pg=analiza-a-modelovani-dynamickych-biologickych-dat--spojite-deterministicke-modely-i>.

10. RACKAUCKAS, Christopher; NIE, Qing. *Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia*. *Journal of Open Research Software*. 2017, roč. 5, č. 1.
11. MA, Yingbo; GOWDA, Shashi; ANANTHARAMAN, Ranjan; LAUGHMAN, Chris; SHAH, Viral; RACKAUCKAS, Chris. *ModelingToolkit: A Composable Graph Transformation System For Equation-Based Modeling*. 2021. Dostupné z arXiv: [2103.05244](https://arxiv.org/abs/2103.05244) [cs.MS].
12. TIBSHIRANI, Robert. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* [online]. 1996, roč. 58, č. 1, s. 267–288 [cit. 2023-03-21]. ISSN 00359246. Dostupné z: <http://www.jstor.org/stable/2346178>.
13. CHLÁDEK, Jakub. *Simulační studie lasso regrese a elastic net regrese v přítomnosti skupinových efektů* [online]. 2021. Dostupné také z: <https://theses.cz/id/hotk48/>. SUPERVISOR: Lukáš Frýd.
14. ZONYGOVÁ, Kristýna. *Restaurace přexponovaného digitálního obrazu* [online]. Brno, 2019 [cit. 2023-03-21]. Dostupné z: <http://hdl.handle.net/11012/179513>. Bakalářská práce. Vysoké učení technické v Brně. Fakulta strojního inženýrství. Ústav matematiky.
15. HRBÁČEK, Radek; RAJMIC, Pavel; VESELÝ, Vítězslav; ŠPIŘÍK, Jan. Řídké reprezentace signálů: úvod do problematiky. *Elektrorevue*. 2011, roč. 2011/50. ISSN 1213-1539. Dostupné také z: <http://www.elektrorevue.cz/cz/clanky/zpracovani-signalu/0/ridke-reprezentace-signalu--uvod-do-problematiky/>.
16. PARIKH, Neal; BOYD, Stephen. Proximal Algorithms. *Found. Trends Optim.* 2014, roč. 1, č. 3, s. 127–239. ISSN 2167-3888. Dostupné z doi: [10.1561/2400000003](https://doi.org/10.1561/2400000003).
17. MANGOVÁ, Marie. *Komprimované snímání v perfuzním zobrazování pomocí magnetické rezonance* [online]. Brno, 2014. Dostupné také z: <https://theses.cz/id/awncx1/>. Diplomová práce. Vysoké učení technické v Brně.
18. ([HTTPS://MATH.STACKEXCHANGE.COM/USERS/98026/SKD](https://math.stackexchange.com/users/98026/skd)), skd. *Derivation of Soft Thresholding Operator / Proximal Operator of L_1 Norm* [Mathematics Stack Exchange]. [B.r.]. Dostupné z eprint: <https://math.stackexchange.com/q/511106>. URL: <https://math.stackexchange.com/q/511106> (version: 2019-02-12).
19. CANDÉS, Emmanuel. *MATH 301: Advanced Topics in Convex Optimization - Lecture 26*. 2015. Dostupné z eprint: <https://candes.su.domains/teaching/math301/Lectures/Consensus.pdf>.
20. MARTENSEN, Julius; RACKAUCKAS, Christopher et al. *DataDrivenDiffEq.jl*. Zenodo, 2021. Dostupné z doi: [10.5281/zenodo.5083412](https://doi.org/10.5281/zenodo.5083412).
21. GUPTA, Ashutosh. *Multiplicative Noise Removal Techniques - A Study*. Dept. of Electrical Engineering, Indian Institute of Science, 2015. Dostupné z doi: [10.13140/RG.2.1.2651.7844/1](https://doi.org/10.13140/RG.2.1.2651.7844/1).

22. BESANÇON, Mathieu; PAPAMARKOU, Theodore; ANTHOFF, David; ARSLAN, Alex; BYRNE, Simon; LIN, Dahua; PEARSON, John. Distributions.jl: Definition and Modeling of Probability Distributions in the JuliaStats Ecosystem. *Journal of Statistical Software*. 2021, roč. 98, č. 16, s. 1–30. ISSN 1548-7660. Dostupné z doi: [10.18637/jss.v098.i16](https://doi.org/10.18637/jss.v098.i16).
23. WIKIPEDIA. *Total variation denoising* – Wikipedia, The Free Encyclopedia. 2023. Dostupné také z: https://en.wikipedia.org/wiki/Total_variation_denoising. [Online; accessed 23-March-2023].
24. CONDAT, Laurent. A Direct Algorithm for 1-D Total Variation Denoising. *IEEE Signal Processing Letters*. 2013, roč. 20, č. 11, s. 1054–1057. Dostupné z doi: [10.1109/LSP.2013.2278339](https://doi.org/10.1109/LSP.2013.2278339).
25. DELEDALLE, Charles-Alban; VAITER, Samuel; PEYRÉ, Gabriel; FADILI, Jallal; DOSSAL, Charles. Unbiased risk estimation for sparse analysis regularization. 2012, s. 3053–3056. Dostupné z doi: [10.1109/ICIP.2012.6467544](https://doi.org/10.1109/ICIP.2012.6467544).
26. WAHLBERG, Bo; BOYD, Stephen; ANNERGREN, Mariette; WANG, Yang. An ADMM Algorithm for a Class of Total Variation Regularized Estimation Problems*. *IFAC Proceedings Volumes*. 2012, roč. 45, č. 16, s. 83–88. ISSN 1474-6670. Dostupné z doi: <https://doi.org/10.3182/20120711-3-BE-2027.00310>. 16th IFAC Symposium on System Identification.
27. FUNDAMENTAL. *TotalVariation.jl* [<https://github.com/fundamental/TotalVariation.jl>]. GitHub, 2013.
28. SELESNICK, Ivan W.; CHEN, Po-Yu. Total variation denoising with overlapping group sparsity. 2013, s. 5696–5700. Dostupné z doi: [10.1109/ICASSP.2013.6638755](https://doi.org/10.1109/ICASSP.2013.6638755).
29. CHARTRAND, Rick. Numerical Differentiation of Noisy, Nonsmooth Data. *ISRN Appl. Math.* 2011, roč. 2011. Dostupné z doi: [10.5402/2011/164564](https://doi.org/10.5402/2011/164564).
30. TIKHONOV, A. N. Regularization of Incorrectly Posed Problems. *Soviet Math. Dokl.* 1963, s. 1624–1627. Dostupné také z: <https://cir.nii.ac.jp/crid/1570291226450870144>.
31. HOROVA, Ivana; ZELINKA, Jiří. *Numerické metody*. 2. vyd. Brno: Masarykova univerzita v Brně, 2004. ISBN 80-210-3317-7.
32. BRUNTON, Steven L.; PROCTOR, Joshua L.; KUTZ, J. Nathan. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*. 2016, roč. 113, č. 15, s. 3932–3937. Dostupné z doi: [10.1073/pnas.1517384113](https://doi.org/10.1073/pnas.1517384113).
33. BRUNTON, Steven L. Sparse Identification of Nonlinear Dynamics (SINDy): Sparse Machine Learning Models 5 Years Later! 2021. Dostupné z doi: [10.52843/cassyni.sx3npx](https://doi.org/10.52843/cassyni.sx3npx).
34. DANIELS, Bryan C.; NEMENMAN, Ilya. Automated adaptive inference of phenomenological dynamical models. *Nature Communications*. 2015, roč. 6, č. 1. Dostupné z doi: [10.1038/ncomms9133](https://doi.org/10.1038/ncomms9133).

35. SCHMIDT, Michael; LIPSON, Hod. Distilling Free-Form Natural Laws from Experimental Data. *Science*. 2009, roč. 324, č. 5923, s. 81–85. Dostupné z doi: [10.1126/science.1165893](https://doi.org/10.1126/science.1165893).
36. HEIM, Niklas; PEVNY, Tomas; SMIDL, Vasek. Neural Power Units. In: LAROCHELLE, H.; RANZATO, M.; HADSELL, R.; BALCAN, M.F.; LIN, H. (ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020, sv. 33, s. 6573–6583. Dostupné také z: https://proceedings.neurips.cc/paper_files/paper/2020/file/48e59000d7dfcf6c1d96ce4a603ed738-Paper.pdf.
37. RUDY, Samuel H.; BRUNTON, Steven L.; PROCTOR, Joshua L.; KUTZ, J. Nathan. *Data-driven discovery of partial differential equations*. 2016. Dostupné z arXiv: [1609.06401 \[nlin.PS\]](https://arxiv.org/abs/1609.06401).
38. CHAMPION, Kathleen; ZHENG, Peng; ARAVKIN, Aleksandr Y.; BRUNTON, Steven L.; KUTZ, J. Nathan. *A unified sparse optimization framework to learn parsimonious physics-informed models from data*. 2020. Dostupné z arXiv: [1906.10612 \[physics.comp-ph\]](https://arxiv.org/abs/1906.10612).
39. BRUNTON, Steven L. Sparse Nonlinear Dynamics Models with SINDy, Part 2: Training Data & Disambiguating Models. 2021. Dostupné z doi: [10.52843/cassyni.y7yb6y](https://doi.org/10.52843/cassyni.y7yb6y).
40. CHAMPION, Kathleen P.; BRUNTON, Steven L.; KUTZ, J. Nathan. Discovery of Nonlinear Multiscale Systems: Sampling Strategies and Embeddings. *SIAM Journal on Applied Dynamical Systems*. 2019, roč. 18, č. 1, s. 312–333. Dostupné z doi: [10.1137/18m1188227](https://doi.org/10.1137/18m1188227).
41. SCHAEFFER, Hayden; MCCALLA, Scott G. Sparse model selection via integral terms. *Phys. Rev. E*. 2017, roč. 96, s. 023302. Dostupné z doi: [10.1103/PhysRevE.96.023302](https://doi.org/10.1103/PhysRevE.96.023302).
42. BRUNTON, Steven L. Sparse Nonlinear Dynamics Models with SINDy, Part 3: Effective Coordinates for Parsimonious Models. 2021. Dostupné z doi: [10.52843/cassyni.5z2jld](https://doi.org/10.52843/cassyni.5z2jld).
43. BAKARJI, Joseph; CHAMPION, Kathleen; KUTZ, J. Nathan; BRUNTON, Steven L. *Discovering Governing Equations from Partial Measurements with Deep Delay Autoencoders*. arXiv, 2022. Dostupné z doi: [10.48550/ARXIV.2201.05136](https://doi.org/10.48550/ARXIV.2201.05136).
44. BRUNTON, Steven L. Sparse Nonlinear Dynamics Models with SINDy, Part 4: The Library of Candidate Nonlinearities. 2021. Dostupné z doi: [10.52843/cassyni.p7t7ds](https://doi.org/10.52843/cassyni.p7t7ds).
45. TROPP, J.A. Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*. 2004, roč. 50, č. 10, s. 2231–2242. Dostupné z doi: [10.1109/TIT.2004.834793](https://doi.org/10.1109/TIT.2004.834793).
46. BRUNTON, Steven L. Sparse Nonlinear Dynamics Models with SINDy, Part 5: The Optimization Algorithms. 2021. Dostupné z doi: [10.52843/cassyni.f1bn05](https://doi.org/10.52843/cassyni.f1bn05).

47. ZHENG, Peng; ASKHAM, Travis; BRUNTON, Steven L.; KUTZ, J. Nathan; ARAVKIN, Aleksandr Y. *A Unified Framework for Sparse Relaxed Regularized Regression: SR3*. 2018. Dostupné z arXiv: [1807.05411](https://arxiv.org/abs/1807.05411) [stat.ML].
48. HODGKIN, A. L.; HUXLEY, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*. 1952, roč. 117, č. 4, s. 500–544. Dostupné z doi: <https://doi.org/10.1113/jphysiol.1952.sp004764>.
49. JEKL, Jan. *Matematické modely neuronu* [online]. 2018 [cit. 2023-04-13]. Dostupné také z: <https://is.muni.cz/th/iibr5/>. Diplomová práce. Masarykova univerzita, Přírodovědecká fakulta, Brno. SUPERVISOR : Ondřej Pokora.
50. GERSTNER, Wulfram; KISTLER, Werner M. *Spiking neuron models*. Cambridge University Press, 2002. Dostupné z doi: <https://doi.org/10.1017/CB09780511815706>.
51. FITZHUGH, Richard. Impulses and Physiological States in Theoretical Models of Nerve Membrane. *Biophysical Journal*. 1961, roč. 1, č. 6, s. 445–466. ISSN 0006-3495. Dostupné z doi: [https://doi.org/10.1016/S0006-3495\(61\)86902-6](https://doi.org/10.1016/S0006-3495(61)86902-6).
52. LORENZ, Edward N. Deterministic nonperiodic flow. *Journal of atmospheric sciences*. 1963, roč. 20, č. 2, s. 130–141.
53. PŘIBYLOVÁ, Lenka. *Teorie bifurkací, chaos a fraktály*. Multimediální elektronický výukový materiál. 1., vyd. Brno: Masarykova univerzita, 2021. ISBN 978-80-280-0112-4. Dostupné také z: https://is.muni.cz/do/rect/el/estud/prif/2020podzim/teorie_bifurkaci/web/index.html.
54. WEISSTEIN, Eric W. *Lorenz Attractor: From MathWorld – A Wolfram Web Resource*. [B.r.]. Dostupné také z: <https://mathworld.wolfram.com/LorenzAttractor.html>. cit. 2023-04-18.
55. WIKIPEDIA CONTRIBUTORS. *Hodgkin–Huxley model* — Wikipedia, The Free Encyclopedia. 2023. Dostupné také z: https://en.wikipedia.org/w/index.php?title=Hodgkin%E2%80%93Huxley_model&oldid=1151623295. [Online; accessed 5-May-2023].
56. WIKISKRIPTA. *Iontové kanály*. 2022. Dostupné také z: https://www.wikiskripta.eu/index.php?title=Iontov%C3%A9_kan%C3%A1ly&oldid=458313. [Online; navštíveno 5. 05. 2023].
57. NELDER, J. A.; MEAD, R. A Simplex Method for Function Minimization. *The Computer Journal*. 1965, roč. 7, č. 4, s. 308–313. ISSN 0010-4620. Dostupné z doi: [10.1093/comjnl/7.4.308](https://doi.org/10.1093/comjnl/7.4.308).
58. MOGENSEN, Patrick Kofod; RISETH, Asbjørn Nilsen. Optim: A mathematical optimization package for Julia. *Journal of Open Source Software*. 2018, roč. 3, č. 24, s. 615. Dostupné z doi: [10.21105/joss.00615](https://doi.org/10.21105/joss.00615).

