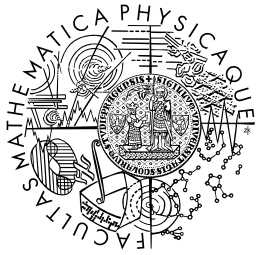


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Pavol Šimkovič

Řešení Zermelova navigačního problému pomocí level-set metody

Matematický ústav UK

Vedoucí bakalářské práce: RNDr. Karel Tůma, Ph.D.

Studijní program: Matematické modelování

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Ďakujem svojmu vedúcemu za jeho nadšenie pre úlohu a dlhé spoločne strávené hodiny nad výpočtami. Tiež ďakujem svojim kolegom za to, že so mnou diskutovali o úlohe a dávali mi inšpiratívne podnety.

Název práce: Řešení Zermelova navigačního problému pomocí level-set metody

Autor: Pavol Šimkovič

Ústav: Matematický ústav UK

Vedoucí bakalářské práce: RNDr. Karel Tůma, Ph.D., Matematický ústav UK

Abstrakt: Efektívne plánovanie trasy pri diaľkových letoch je kriticky dôležité, ak vzdušné prúdy výrazne ovplyvňujú pohyb plavidla, ako tomu je v prípade lode alebo vzducholode. Optimálne riadenie môže ušetriť čas cesty, spotrebu paliva a včasným varovaním zabrániť letu v nepriaznivých podmienkach. Implementovali sme numerický algoritmus na výpočet časovo optimálnej trajektórie pre diaľkový let vzducholode v premenlivom atmosférickom prúdení. Testovali sme ho na modelových zadaniach aj reálnych meteorologických dátach. Výsledkom práce je softvér s grafickým rozhraním schopný vypočítať optimálne navigačné inštrukcie, získať automatizovane predpoveď počasia a vykresliť výstupy, ktorých súčasťou je aj indikácia presnosti výpočtu.

Klíčová slova: vzducholoď, letecká navigácia, optimálne riadenie, Zermelova úloha, levelset, fast marching method, Djikstrov algoritmus

Title: Solution of Zermelo's navigation problem using level-set method

Author: Pavol Šimkovič

Institute: Mathematical Institute of Charles University

Supervisor: RNDr. Karel Tůma, Ph.D., Mathematical Institute of Charles University

Abstract: Effective path planning for long-distance flights is critically important when air streams significantly affect airship movement, as is the case for a ship or a zeppelin. Optimal control can save travel time, fuel consumed and avoid flying in unfavourable conditions by early warning. We implemented a numerical algorithm for computing time-optimal trajectory for a long-distance flight of a zeppelin in variable atmospheric flow. We tested the algorithm on both model tasks and real meteorological data. The result of our work is a software with graphical interface, which can compute optimal navigation instructions, obtain weather forecast automatically and plot outputs, which contain also an indication of computation precision.

Keywords: zeppelin, airship navigation, optimal control, Zermelo's problem, levelset, fast marching method, Djikstra's algorithm

Obsah

Úvod

Efektívne plánovanie trasy má dôležitú úlohu pri navigácii plavidiel. Hlavná uvažovaná aplikácia spočíva v diaľkových letoch. Môžeme optimalizovať trvanie cesty, spotrebu paliva alebo prihliadať na bezpečnosť plavidla. V prípade, že vplyv prúdenia na pohyb plavidla nie je zanedbateľný, je hľadanie optimálneho riadenia netriviálny problém. Potreba spoľahlivého algoritmu na počítanie navigačných inštrukcií je zrejmá.

V tejto práci riešime realistický problém navigácie vzducholode v nestacionárnom atmosférickom prúdení, ktorým je vzducholoď unášaná. Vzducholoď sa môže pohybovať vlastným pohonom s obmedzenou rýchlosťou. Cieľom je dostať sa z východzieho bodu do cieľového v najkratšom čase. Výstupom bude sada navigačných inštrukcií pre vlastný pohon vzducholode, ako vedľajší produkt výpočtu získame trasu a čas cesty. Ide o príklad reálnej situácie, v ktorej efektívne využitie prúdenia je pre splnenie úlohy kritické.

Popísaný navigačný problém nachádza uplatnenie nielen v leteckej doprave, ale aj v morskej a podmorskej navigácii alebo pri riadení automatizovaných sond. Dosiahnuté výsledky sú priamo aplikovateľné aj pre diaľkovú plavbu po mori. Pre jednotnosť textu však budeme v celej práci hovoriť o vzducholodi.

Náš prístup ku tejto komplexnej úlohe je po teoretickej stránke jednoduchý, aj keď nie je priamy. Pri jeho numerickej implementácii sa však vynárajú rôzne problémy. Ich riešenie vyžaduje techniky, ktoré sú výrazne zložitejšie ako teoretický rámec úlohy. Používame level-set metódu, ktorú popíšeme v druhej kapitole práce. Je to robustný spôsob riešenia, ktorý na druhú stranu zvyšuje výpočtové nároky, keďže pridáva problému dimenziu navyše. Predstavíme aplikáciu level-set metódy na náš problém a uvidíme, že výpočet riešenia prebieha v niekoľkých fázach.

V tretej kapitole predstavíme použité numerické techniky. Popíšeme realizáciu jednotlivých fáz výpočtu, numerické problémy, ktoré vznikajú a spôsoby, akými ich riešime. Uvedieme tiež diskusiu alternatívnych prístupov k numerickému riešeniu.

Vytvoríme plne funkčný riešič navigačnej úlohy. Vyvinutý program testujeme na akademických príkladoch. Na konkrétnej úlohe so stacionárnym prúdením ho porovnáme s alternatívnym prístupom, ktorý využíva nemennosť prúdenia v čase a je preto presnejší. Tým otestujeme presnosť nášho riešiča. Následne testujeme riešič na realistických zadaniach. Popri tom vyvinieme grafické rozhranie s automatizovaným získavaním predpovede počasia a intuitívnym ovládaním. Výsledky uvádzame vo štvrtej kapitole.

Na záver zhrnieme dosiahnuté výsledky a diskutujeme kvalitu a spoľahlivosť vytvoreného programu nielen po stránke numerického výpočtu, ale aj z hľadiska použitých predpokladov a aplikovateľnosti v skutočnom svete. Tiež uvedieme širší kontext riešenej úlohy a použiteľnosti výsledkov práce.

1 Formulácia úlohy

Riešime realistický problém navigácie plavidla pohybujúceho sa v pohyblivom spojitom prostredí ovplyvňujúcom jeho rýchlosť. Plavidlo sa dokáže pohybovať vlastným pohonom, ktorým vie riadiť svoj smer, avšak má obmedzenú maximálnu rýchlosť. Zároveň je unášané prostredím, ktorého rýchlostné pole je ľubovoľné. Cieľom plavidla je dostať sa z východzieho bodu do cieľového po optimálnej trajektórii, pričom optimalita trajektórie môže byť definovaná rôzne. Teda úlohou je zistiť časový priebeh veľkosti a smeru vlastného pohonu plavidla tak, aby sa splnili zadané kritériá optimality. V našej práci uvažujeme diaľkový let vzducholode, resp. diaľkovú plavbu lode v oceáne.

Trajektória vzducholode môže byť optimalizovaná pre minimálny čas cesty, minimálnu spotrebu paliva, môže zohľadňovať bezpečnosť vzducholode (napr. vzhľadom na turbulentné resp. prudko premenlivé prúdenie alebo zakázané resp. rizikové zóny) alebo kombináciu spomenutých faktorov. Pri časovo závislom prúdení môže byť prospešné určiť optimálny čas štartu vzducholode (vo vymedzenom rozsahu) tak, aby boli najlepšie splnené ostatné kritériá. V našom prípade prichádzajú do úvahy aj medzipristátia z dôvodu nepriaznivých podmienok, v niektorých krajinách členitosť terénu alebo prípadné prekročenie štátnej hranice (vrátane vnútroštátnych letov, ak to vynucuje charakter prúdenia).

1.1 Zjednodušený model

Vzhľadom k tomu, že riešime úlohu pre diaľkový let, sú prijateľné nasledovné zjednodušenia. Uvažovaná priestorová škála prúdenia je omnoho väčšia ako rozmery vzducholode. Rýchlosť zmien prúdenia je omnoho menšia ako schopnosť vzducholode zmeniť rýchlosť a smer vlastného pohonu. Z toho vyplývajú nasledovné predpoklady.

V modeli neuvažujeme zotrvačnosť ani moment zotrvačnosti vzducholode. Vzducholoď vie zmeniť veľkosť a smer vlastného pohonu ľubovoľne rýchlo v rámci daných obmedzení. Interakcia vzducholode a prostredia je okamžitá a izotropná. Neuvažujeme teda napr. závislosť odporu vzduchu na natočení vzducholode vzhľadom na prúdenie. Vlastný pohon vzducholode určuje relatívnu rýchlosť vzducholode vzhľadom na prostredie. Celková rýchlosť vzducholode je teda súčet vlastného pohonu a rýchlosti prúdenia v danom mieste.

Optimalizujeme čas letu vzducholode. Výstupom nášho riešenia bude taká sada inštrukcií pre smer a natočenie vlastného pohonu vzducholode, ktorou sa minimalizuje čas cesty z východzieho bodu do cieľového. Ostatné spomenuté kritériá neuvažujeme. Vzducholoď začína cestu v pevnom zadanom čase a letí nepretržite, dokým dosiahne cieľ.

Úlohu riešime v dvoch priestorových rozmeroch, teda prúdenie považujeme za nezávislé na vertikálnej súradnici.

Na prúdenie nekladieme žiadne ďalšie obmedzenia. Navrhnutá metóda dokáže vyriešiť úlohu pre ľubovoľné realistické prúdenie, ktoré môže byť závislé na čase. Môžeme dokonca zadať a úlohu vyriešiť pre nespojité prúdenie, to ale bude v riešiči aproximované spojitou funkciou.

1.2 Zermelov navigačný problém

Úloha, ktorú riešime, je tiež známa ako Zermelov navigačný problém, ktorý formuloval Ernst Zermelo v roku 1931. Zermelova formulácia znie [1]:

> V neohraničenej rovine, v ktorej je rozloženie vetra dané vektorovým poľom ako funkciou polohy a času, sa pohybuje plavidlo konštantnou rýchlosťou relatívnou k okolitej vzduchovej hmote. Ako má byť plavidlo riadené, aby sa dostalo zo štartovného bodu do zadaného cieľa v najkratšom čase?

Môžeme si všimnúť, že Zermelo neuvažoval, že by sa plavidlo pohybovalo pomalšie, ako je jeho maximálna rýchlosť. Ako uvidíme neskôr, tento predpoklad je bez ujmy na všeobecnosti.

2 Matematický model

Úlohu riešime v oblasti $\Omega \subset \mathbb{R}^2$. Máme zadanú predpoveď počasia – smer a rýchlosť vetra v krajine – ako dvojrozmerné rýchlostné pole:

$$\begin{aligned} \vec{V} : \ \Omega \times [0, +\infty) &\to \Omega \times \mathbb{R}^2 \\ (\vec{x},t) &\mapsto \vec{V}(\vec{x},t). \end{aligned}$$

Ďalej poznáme štartový bod $\vec{x}_s \in \Omega$, v ktorom vzducholoď začína v čase $t_0 = 0$, a cieľový bod $\vec{x}_f \in \Omega$, do ktorého sa vzducholoď potrebuje dostať. Zadaná je maximálna rýchlosť vzducholode vzhľadom na prostredie $F_{\max} \in \mathbb{R}^+$. Je tiež možné (a numericky potrebné) zadať najdlhší prípustný čas cesty $T_{\max} \in \mathbb{R}^+$.

Označme $T(\vec{x}) \in [0, +\infty]$ najskorší čas, kedy možno dosiahnuť bod $\vec{x} \in \Omega$ (začínajúc v $\vec{x}_s$). Ak $T(\vec{x}) = +\infty$, znamená to, že vzducholoď nedokáže daný bod dosiahnuť v žiadnom čase. Ďalej označme $T_f := T(\vec{x}_f)$.

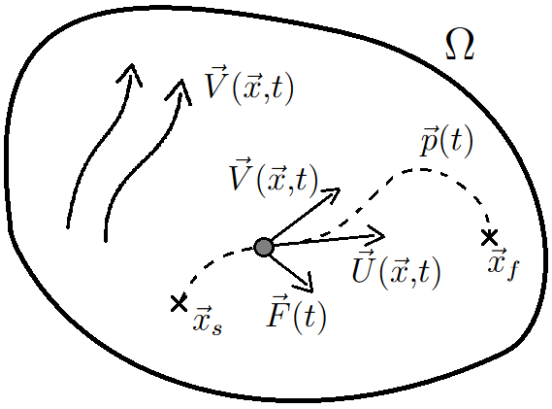
Označme ďalej $\vec{p}(t) \in \Omega$ trajektóriu vzducholode a $\vec{F}(t)$ vlastnú rýchlosť vzducholode s rozkladom na veľkosť a smer

$$\vec{F}(t) = F(t)\vec{h}(t), \tag{2.1}$$

kde $F(t)$ je veľkosť rýchlosti a $\vec{h}(t)$ je jednotkový vektor v $\mathbb{R}^2$. Funkcie $\vec{p}$, $\vec{F}$, F a $\vec{h}$ sú definované pre $t \in [0,T(\vec{x}_f)]$.

Skutočnú rýchlosť vzducholode označíme $\vec{U}(\vec{x},t)$, viď obrázok 2.1. Zrejme platí

$$\vec{U}(\vec{x},t) = \vec{V}(\vec{x},t) + \vec{F}(t). \tag{2.2}$$



Obr. 2.1 Základné značenie v práci.

V tejto práci implementujeme numerický algoritmus, ktorý rieši Zermelov navigačný problém popísaný v časti 1.1.

Výstupom algoritmu bude trvanie cesty, navigačné inštrukcie, trasa a prípadne informácia o neriešiteľnosti úlohy alebo možnej veľkej nepresnosti spočítaného riešenia. Trvaním cesty rozumieme čas $T_f \in \mathbb{R}^+$ potrebný na dosiahnutie bodu $\vec{x}_f$. Inštrukcie predstavuje funkcia $\vec{F} : \ [0,T_f] \to \mathbb{R}^2$ určujúca natočenie a vlastnú rýchlosť vzducholode v čase. Trasa je funkcia $\vec{p}(t) : \ [0,T_f] \to \Omega$ určujúca polohu

vzducholode v čase. Neriešiteľnosť úlohy môže znamenať napríklad, že $T_f > T_{\max}$, podrobnosti uvidíme v časti o numerickej implementácii.

V kontexte zavedeného značenia môžeme formulovať nasledovné predpoklady:

- vzducholoď predstavuje (pohyblivý) bod v množine Ω, je unášaná prostredím a jej rýchlosť je daná vzťahom (2.2) (zanedbali sme zotrvačnosť a rozmery vzducholode),
- trajektória vzducholode je spojitá, t.j. $\vec{p} \in C[0,T_f]$,
- vzducholoď nemôže opustiť oblasť Ω (podrobnosti viď časť o numerickej implementácii).

Zrejmé sú nasledovné pozorovania:

- $F(t) \leq F_{\max} \qquad \forall t \in [0,T_{\max}]$,
- $|\vec{h}(t)| = 1 \qquad \forall t \in [0,T_{\max}]$,
- $T(\vec{x}_s) = 0$,
- $\vec{p}(0) = \vec{x}_s$,
- $\vec{p}(T_f) = \vec{x}_f$,
- T je definované na celej Ω,
- $\vec{p}([0,T_f])$ je v prípade, že riešenie úlohy existuje a $\vec{p}$ predstavuje optimálnu trajektóriu, jednorozmerná podmnožina Ω.

Pre trajektóriu zrejme platí:

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{p}(t) = \vec{U}(\vec{p}(t),t) = \vec{V}(\vec{p}(t),t) + F(t)\vec{h}(t). \tag{2.3}$$

Ďalej môžeme formulovať zadanie v kontexte zavedeného značenia nasledovne: Hľadáme takú sadu inštrukcií $\vec{F}(t)$, ktorá minimalizuje čas T_f za predpokladu, že pohyb vzducholode je určený vzťahom (2.3).

2.1 Hlavná myšlienka riešenia

Možných sád inštrukcií pre riadenie vzducholode je toľko, koľko je reálnych funkcií $[0,T_{\max}] \to [0,F_{\max}] \times [0,2\pi)$. Priamočiara optimalizácia na množine týchto funkcií neprichádza do úvahy vzhľadom k takmer žiadnym obmedzeniam na veterné pole.

Je potrebné abstrahovať od konkrétnych trajektórií. Lolla vo svojej práci [2] predstavil postup, akým možno problém redukovať do podoby vhodnej pre numerickú implementáciu.

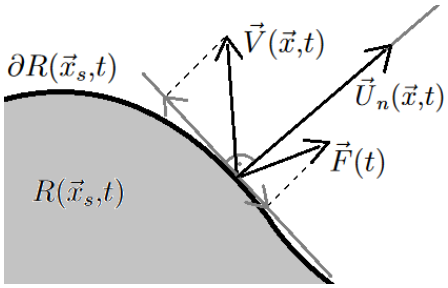
Zaveďme množinu dosiahnuteľnosti podobne, ako je zavedená v práci [2]:

$$R(\vec{x}_s,t) = \{\vec{x} \in \Omega \mid \exists \text{ trajektória } \vec{p}_{\vec{x}} : \vec{p}_{\vec{x}}(0) = \vec{x}_s \text{ a } \vec{p}_{\vec{x}}(t) = \vec{x}\}.$$

Množina dosiahnuteľnosti v čase t teda predstavuje množinu všetkých bodov, kam sa môže vzducholoď dostať z daného štartovného bodu v čase t. Nemusí nutne obsahovať body, cez ktoré vzducholoď prešla v minulosti, ak je vietor príliš silný. Ďalej frontom dosiahnuteľnosti budeme rozumieť hranicu tejto množiny.

Front dosiahnuteľnosti predstavuje v istom zmysle množinu najvzdialenejších bodov, ktoré možno dosiahnuť pri optimálnom riadení. Zároveň neobsahuje informácie o konkrétnych trajektóriách.

Budeme riešiť časový vývoj frontu dosiahnuteľnosti. Myšlienkou riešenia tejto úlohy je, že pri optimálnom riadení vzducholoď (nachádzajúca sa na fronte dosiahnuteľnosti) smeruje svoj vlastný pohon v smere vonkajšej normály k tomuto frontu maximálnou rýchlosťou. Intuitívne táto voľba dáva zmysel. Vzducholoď nachádzajúca sa na fronte má možnosť sa pohybovať vlastným pohonom ľubovoľným smerom a rýchlosťou nanajvýš $F_{\max}$. Front predstavuje optimálnu množinu, teda pri jeho evolúcii vyžadujeme, aby vzducholoď volila svoju rýchlosť tak, aby sa front posúval vpred čo najrýchlejšie. Celková rýchlosť vzducholode je súčtom vlastného pohonu a vetra v danom mieste. Rozdeľme túto rýchlosť na normálovú a tangenciálnu zložku. Tangenciálna zložka neprispieva k pohybu frontu, pretože vzducholoď ostáva vo fronte pre daný fixný čas a rýchlosť pohybu frontu v danom mieste je teda určená len normálovou zložkou. Viď obrázok 2.2. To znamená, že chceme maximalizovať normálovú zložku celkovej rýchlosti vzducholode. Zrejme toho dosiahneme tak, že budeme vzducholoď riadiť najvyššou možnou rýchlosťou po smere normály k frontu. Exaktnejšiu formuláciu uvidíme v nasledovných častiach.



Obr. 2.2 Myšlienka riešenia spočíva v tom, že na pohyb frontu v každom jeho bode vplýva len zložka celkovej rýchlosti vzducholode, ktorá je normálová k frontu v danom bode, teda $\vec{U}_n(\vec{x},t)$. Zložka k nej kolmá predstavuje pohyb vzducholode v rámci frontu a neprispieva k jeho pohybu. Normálovú zložku maximalizujeme tak, že rýchlosť vzducholode bude najvyššia možná a bude mať normálový smer.

Akonáhle front dosiahne cieľ, vieme, že existuje trajektória, ktorou sa v danom čase vzducholoď dostane do cieľa. Keďže front predstavuje v istom zmysle najlepší výsledok optimálneho riadenia, ak front ešte nedosiahol cieľ, neexistuje spôsob riadenia vzducholode, ktorým by sa vzducholoď v tomto čase dostala do cieľa. Vývoj frontu budeme počítať nanajvýš do zadaného času $T_{\max}$ a ak dovtedy front nedosiahne cieľ, prehlásime, že riešenie neexistuje.

V tejto chvíli prestávame počítať vývoj frontu. Poznáme čas cesty a front dosiahnuteľnosti v každom čase $t \leq T_f$. Front síce neobsahuje informáciu o konkrétnej trajektórii, ale vieme, že v čase T_f sa vzducholoď nachádza v bode $\vec{x}_f$. Trajektória vzducholode je určená vzťahom (2.3), pričom podľa diskusie vyššie vieme určiť $\vec{F}(t)$ z frontu v čase t. Tieto informácie nám umožňujú spočítať optimálnu trajektóriu aj inštrukcie pre vzducholoď spätnou integráciou vzťahu (2.3).

2.2 Metóda úrovňových množín (level-set)

Je potrebné nájsť numericky prijateľný spôsob, ako riešiť časový vývoj frontu. Jedná sa tu o problém sledovania rozhrania, ktorý sa štandardne rieši metódou úrovňových množín (level-set). Podobne ako v práci [2] zavádzame funkciu

$$\begin{aligned}\phi : \Omega \times \mathbb{R}^+ &\to \mathbb{R},\\ (\vec{x},t) &\mapsto \phi(\vec{x},t).\end{aligned}$$

Úrovňovou množinou pre $C \in \mathbb{R}$ v čase $t \geq 0$ označme množinu

$$\Gamma(C,t) := \{\vec{x} \in \Omega \mid \phi(\vec{x},t) = C\}.$$

Rozhranie (front dosiahnuteľnosti) bude reprezentované nulovou úrovňovou množinou $\Gamma_0(t) := \Gamma(0,t)$. Funkciu úrovňovej množiny $\phi(\vec{x},t)$ budeme ďalej skrátene nazývať levelset.

Odvodíme evolučnú rovnicu levelsetu. Nech $C \in \mathbb{R}$ je ľubovoľné a pre úrovňovú množinu platí

$$\phi(\vec{x},t) = C.$$

Tento vzťah derivujeme totálne podľa času a dostávame

$$\frac{\partial}{\partial t}\phi(\vec{x},t) + \sum_i \frac{\partial \phi}{\partial x_i}\frac{\mathrm{d}x_i}{\mathrm{d}t} = 0,$$

čo možno upraviť na

$$\frac{\partial}{\partial t}\phi(\vec{x},t) + \vec{\nabla}\phi \cdot \frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = 0.$$

Odtiaľ máme výslednú evolučnú rovnicu

$$\frac{\partial}{\partial t}\phi(\vec{x},t) = -\vec{\nabla}\phi \cdot \frac{\mathrm{d}\vec{x}}{\mathrm{d}t}. \tag{2.4}$$

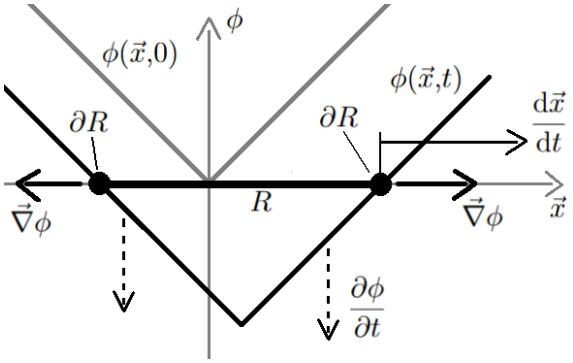
V tejto rovnici výraz $\frac{\mathrm{d}\vec{x}}{\mathrm{d}t}$ predstavuje rýchlostné pole (rýchlosť pohybu rozhrania) v danom mieste a čase.

2.3 Aplikácia level-set metódy na Zermelov navigačný problém

Aplikujeme odvodenú metódu na náš problém. Pre matematický popis budeme požadovať trochu silnejšie predpoklady, ako sme formulovali pre reálnu úlohu na začiatku tejto kapitoly. Budeme požadovať pre trajektóriu existenciu a spojitosť $\frac{\mathrm{d}\vec{p}}{\mathrm{d}t}$. Levelset, ktorý sme zaviedli v predošlej časti, použijeme na sledovanie frontu dosiahnuteľnosti a budeme požadovať $\phi \in C^1(\Omega \times \mathbb{R}_0^+)$. Front bude predstavovať nulovú úrovňovú množinu:

$$\Gamma_0(t) = \partial R(\vec{x}_s,t).$$

Rýchlosť vzducholode v danom mieste a čase ($\vec{U}(\vec{x},t)$) bude pre levelset predstavovať rýchlostné pole $\frac{\mathrm{d}\vec{x}}{\mathrm{d}t}$. Všimnime si, že pre úlohu je podstatné správanie levelsetu len v okolí Γ_0.



Obr. 2.3 Názorná ilustrácia evolúcie levelsetu a frontu. Na horizontálnej osi je priestorová súradnica, na vertikálnej osi hodnota levelsetu.

Na fronte je $\phi(\vec{x},t) = 0$, ďalej zaveďme konvenciu:

$$\begin{aligned} \phi(\vec{x},t) > 0 \qquad & \text{mimo } R(\vec{x}_s,t) \\ \phi(\vec{x},t) < 0 \qquad & \text{vnútri } R(\vec{x}_s,t). \end{aligned}$$

Vonkajšiu normálu k množine dosiahnuteľnosti možno určiť ako $\frac{\vec{\nabla}\phi}{|\vec{\nabla}\phi|}$. Tento výraz je definovaný aj mimo front (pokiaľ tam existuje gradient levelsetu), čo budeme ďalej využívať. Vývoj levelsetu je ilustrovaný na obr. 2.3.

Za rýchlostné pole dosadíme rýchlosť vzducholode, ako sme ju popísali v časti 2.1:

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{V}(\vec{x},t) + \vec{F}(t) = \vec{V}(\vec{x},t) + F_{\max}\vec{n} = \vec{V}(\vec{x},t) + F_{\max}\frac{\vec{\nabla}\phi}{|\vec{\nabla}\phi|}.$$

V prvej rovnosti sme dosadili rýchlosť vzducholode z (2.2), ďalej sme použili výsledok úvah v časti 2.1 a v poslednej rovnosti sme vyjadrili normálu z gradientu levelsetu.

Dosadením získanej rovnosti do rovnice (2.4) dostávame

$$\frac{\partial\phi}{\partial t} = -\vec{\nabla}\phi \cdot \left(F_{\max}\frac{\vec{\nabla}\phi}{|\vec{\nabla}\phi|} + \vec{V}(\vec{x},t) \right).$$

Po malej úprave získavame evolučnú rovnicu pre levelset:

$$\frac{\partial\phi}{\partial t} + \vec{\nabla}\phi \cdot \vec{V}(\vec{x},t) + F_{\max}|\vec{\nabla}\phi| = 0. \tag{2.5}$$

Rovnicu možno použiť na určenie vývoja levelsetu aj mimo front, pritom sme ju odvodili z požadovaného správania študovaného systému na fronte. Keďže pre úlohu je podstatné správanie levelsetu len na okolí frontu, voľba hodnôt ϕ mimo front je ľubovoľná a to využijeme na zlepšenie numerických vlastností postupu riešenia. Z podstaty úlohy je počiatočná podmienka pre front $R(\vec{x}_s,0) = \{\vec{x}_s\}$. Ako počiatočnú podmienku pre levelset volíme v súlade s konvenciou $\phi(\vec{x},0) = |\vec{x} - \vec{x}_s|$.

Nasledovná veta prevzatá z [2] ukazuje platnosť práve odvodenej rovnice pre riešenie Zermelovej úlohy.

Veta. *Nech $T(\vec{y})$ je najskorší čas príchodu v bode $\vec{y} \in \Omega$ s počiatočnou polohou $\vec{x}_s = 0$ v čase $t = 0$. Vzducholoď sa pohybuje v externom poli $\vec{V}(\vec{x},t)$ rýchlosťou nanajvýš $F_{\max}$. Funkcia $\phi(\vec{x},t)$ je lipschitzovsky spojitá, $\phi : \Omega \times [0, +\infty) \to \mathbb{R}$ a je daná rovnicou* (2.5)*:*

$$\frac{\partial}{\partial t}\phi(\vec{x},t) + \vec{\nabla}\phi(\vec{x},t) \cdot \vec{V}(\vec{x},t) + F_{\max}|\vec{\nabla}\phi(\vec{x},t)| = 0$$

s počiatočnou podmienkou $\phi(\vec{x},0) = ||\vec{x}||_2$. Potom:

(i) $\phi(\vec{y},T(\vec{y})) = 0$,

(ii) neexistuje $t < T(\vec{y})$ *také, že* $\phi(\vec{y},t) = 0$ *a optimálne smerovanie kormidla je* $\vec{h}(t) = \frac{\vec{\nabla}\phi(\vec{y},t)}{|\vec{\nabla}\phi(\vec{y},t)|}$,

(iii) optimálna trajektória $\vec{p}(t)$ sa získa riešením rovnice

$$\frac{\mathrm{d}\vec{p}}{\mathrm{d}t} = \vec{V}(\vec{p},t) + F_{\max}\frac{\vec{\nabla}\phi(\vec{p},t)}{|\vec{\nabla}\phi(\vec{p},t)|} \tag{2.6}$$

s počiatočnou podmienkou $\vec{p}(T(\vec{y})) = \vec{p}(T_f) = \vec{y}$, naspäť v čase.

Dôkaz. Možno nájsť v [2], s. 77–86. □

Body (i) a (ii) tejto vety sú ekvivalentné výsledkom úvah z časti 2.1. V bode (iii) je uvedený vzorec, ktorý sa získa dosadením riadenia vzducholode podľa odvodeného princípu do rovnice (2.3).

2.4 Reinicializácia

Keďže nezáleží na tom, aké hodnoty nadobúda levelset mimo front, môžeme zvoliť jeho hodnoty tak, aby sme zlepšili numerické správanie modelu a dosiahli ďalších príjemných vlastností. Ako počiatočnú podmienku volíme najjednoduchšiu funkciu spĺňajúcu počiatočnú polohu frontu $R(\vec{x}_s,0) = \{\vec{x}_s\}$ a znamienkovú konvenciu: vzdialenosť od štartu,

$$\phi(\vec{x},0) = |\vec{x} - \vec{x}_s|.$$

Ďalej budeme chcieť, aby v ľubovoľnom kroku ϕ predstavovala znamienkovú vzdialenosť od frontu:

$$\phi(\vec{x},t) = \operatorname{sign}(\vec{x}) \min_{\vec{y} \in \Gamma_0(t)} |\vec{x} - \vec{y}| \qquad \forall \vec{x} \in \Omega. \tag{2.7}$$

Znamienko $\operatorname{sign}(\vec{x})$ tu definujeme v súlade so znamienkovou konvenciou nasledovne:

$$\operatorname{sign}(\vec{x}) := \begin{cases} -1 & \text{ak } \vec{x} \in \operatorname{Int} R(\vec{x}_s,t), \\ 0 & \text{ak } \vec{x} \in \partial R(\vec{x}_s,t), \\ +1 & \text{ak } \vec{x} \in \Omega \setminus \overline{R(\vec{x}_s,t)}. \end{cases}$$

Táto podmienka je zrejme splnená pre ϕ v čase 0. V priebehu vývoja levelsetu sa však narúša, a to nielen kvôli numerickým chybám, ale jej nezachovanie plynie aj zo samotnej rovnice (2.5), ako je ukázané v [2], s. 95.

Keďže rovnica, ktorú riešime, prirodzene nezachováva požadovanú vlastnosť levelsetu, budeme raz za čas umelo prepisovať jeho hodnoty tak, aby sa zachoval front (teda nesmieme pri tom zmeniť množinu $\Gamma_0(t)$) a všade mimo front bola splnená podmienka (2.7). Tento proces sa nazýva reinicializácia.

2.5 Redukcia na sústavu ODR pre vietor nezávislý na čase

Ak sa veterné pole v čase nemení, teda $\vec{V}(\vec{x},t) = \vec{V}(\vec{x}) = (V_1(\vec{x}),V_2(\vec{x}))$, je možné problém redukovať na sústavu obyčajných diferenciálnych rovníc. Predpokladáme naviac, že vzducholoď sa celý čas pohybuje maximálnou rýchlosťou (ako sme videli vyššie, na optimalitu nájdeného riešenia to nemá vplyv). Použitím výsledkov z [1] a [3] a metód variačnej analýzy je možné získať nasledovnú sústavu:

$$\begin{aligned}
\frac{\mathrm{d}x_1}{\mathrm{d}t} &= F_{\max}\cos\beta + V_1(\vec{x}),\\
\frac{\mathrm{d}x_2}{\mathrm{d}t} &= F_{\max}\sin\beta + V_2(\vec{x}),\\
\frac{\mathrm{d}\beta}{\mathrm{d}t} &= \frac{\mathrm{d}V_2}{\mathrm{d}x_1}(\vec{x})\sin^2\beta + \left(\frac{\mathrm{d}V_1}{\mathrm{d}x_1}(\vec{x}) - \frac{\mathrm{d}V_2}{\mathrm{d}x_2}(\vec{x})\right)\sin\beta\cos\beta - \frac{\mathrm{d}V_1}{\mathrm{d}x_2}(\vec{x})\cos^2\beta.
\end{aligned} \tag{2.8}$$

Tu, $\vec{x}(t) = (x_1(t),x_2(t))$ predstavuje optimálnu trajektóriu a $\beta(t)$ inštrukcie pre natočenie vzducholode (uhol natočenia). Pre x_1 a x_2 máme počiatočnú podmienku a naviac cieľový bod (bez znalosti času, kedy ho dosiahneme). Pre β však nemáme počiatočnú podmienku. Preto túto sústavu môžeme riešiť napríklad metódou streľby. Skúšame rôzne hodnoty počiatočnej podmienky pre β a riešime numericky sústavu rovníc. Aplikovaním vhodnej numerickej techniky na spresňovanie odhadu počiatočnej podmienky vieme dosiahnuť konvergenciu k optimálnemu riadeniu.

2.6 Záverečné poznámky

Dosiahnutie cieľa vieme zistiť jednoducho vďaka (2.7). Cieľ sme dosiahli, akonáhle $\phi(\vec{x}_f,t) \leq 0$, naviac hodnota $\phi(\vec{x}_f,t)$ nám dovtedy určuje (z definície) vzdialenosť frontu od cieľa. Z nej však nemožno usudzovať nič o čase, kedy dosiahneme cieľ.

Zavedením funkcie ϕ sme zvýšili dimenziu úlohy o 1 a tým nevyhnutne vzrástli pamäťové aj časové nároky výpočtu. Keďže hodnoty ϕ ďaleko od frontu nie sú pre úlohu potrebné, je možné riešiť rovnicu (2.5) na (pohyblivom) úzkom páse v okolí frontu, čím sa zníži časová a pamäťová náročnosť výpočtu. V prvej fáze by sa neukladal celý levelset, ale len jeho hodnoty v okolí frontu – stačilo by uložiť len polohu frontu a prípadne normálový smer. Túto techniku sme v práci nepoužili.

Ponúka sa tiež myšlienka neukladať celý front, ale len jeho potrebnú časť z hľadiska optimálnej trajektórie. Dokázalo by to ďalej znížiť pamäťové nároky výpočtu. Tento prístup ale nikam nevedie, nakoľko front neobsahuje informáciu o konkrétnych trajektóriách a v závislosti na veternom poli môže optimálna trajektória viesť ľubovoľným bodom frontu v danom čase. To, kadiaľ vedie optimálna

trajektória, sa dozvieme až v druhej fáze výpočtu, na ktorý potrebujeme mať uloženú pozíciu frontu v každom čase. Naviac, v druhej fáze používame uložené polohy frontu v opačnom poradí, v akom sme ich získali v prvej fáze. Preto nie je možné paralelne počítať prvú a druhú fázu a vyhnúť sa tak ukladaniu levelsetu v každom čase.

2.7 Zhrnutie

Navigačnú úlohu riešime v niekoľkých fázach. Zaviedli sme front dosiahnuteľnosti, ktorého časový vývoj vieme popísať parciálnou diferenciálnou rovnicou (2.5):

$$\frac{\partial \phi}{\partial t} + \vec{\nabla}\phi \cdot \vec{V}(\vec{x},t) + F_{\max}|\vec{\nabla}\phi| = 0$$

pomocou level-set metódy. V prvej fáze riešime vývoj frontu, dokým front nedosiahne cieľový bod $\vec{x}_f$, alebo do stanoveného maximálneho času $T_{\max}$. Priebežne vykonávame reinicializáciu levelsetu, aby bola splnená podmienka (2.7):

$$\phi(\vec{x},t) = \operatorname{sign}(\vec{x}) \min_{\vec{y}\in\Gamma_0(t)} |\vec{x} - \vec{y}| \qquad \forall \vec{x} \in \Omega.$$

V druhej fáze spätne integrujeme rovnicu (2.6):

$$\frac{\mathrm{d}\vec{p}}{\mathrm{d}t} = \vec{V}(\vec{p},t) + F_{\max}\frac{\vec{\nabla}\phi(\vec{p},t)}{|\vec{\nabla}\phi(\vec{p},t)|}$$

do času $t = 0$.

3 Numerické metódy

Úlohu riešime vo dvoch fázach. Prvou fázou je propagácia frontu dosiahnuteľnosti. Front dosiahnuteľnosti sledujeme levelset metódou. Riešime nelineárnu parciálnu diferenciálnu rovnicu (2.5). Ukladáme levelset v každom časovom kroku t_n. Prvá fáza končí, akonáhle front dosiahne cieľový bod $\vec{x}_f$. Potom nastáva druhá fáza, ktorou je spätná integrácia a rekonštrukcia trasy a inštrukcií. Riešime rovnicu (2.6) spätne v čase.

Na riešenie prvej fázy použijeme metódu konečných prvkov implementovanú v riešiči Firedrake, ktorý funguje ako knižnica pre Python. Na riešenie druhej fázy a väčšiny čiastkových numerických problémov si vystačíme so štandardnou knižnicou Pythonu.

3.1 Metóda konečných prvkov

Evolučnú rovnicu levelsetu (2.5) pre použitie metódy konečných prvkov prevedieme do slabej formulácie[1]. Zároveň upresníme požiadavky na kvalitu funkcií vystupujúcich v rovniciach. Pre zadaný vietor budeme požadovať obmedzenosť, teda $\vec{V}(\cdot,t) \in \mathrm{L}^\infty(\Omega)$ pre s.v. $t > 0$. Pre levelset budeme požadovať:

$$\phi \in \mathrm{L}^2\Big((0,T_{\max}); \mathrm{W}^{1,2}(\Omega)\Big) \cap \mathrm{W}^{1,2}\Big((0,T_{\max}); \mathrm{L}^2(\Omega)\Big),$$
$$\lim_{t\to 0} ||\phi(\cdot,t) - \phi(\cdot,0)||_1 = 0.$$

Tieto požiadavky sme stanovili preto, aby slabá formulácia dávala zmysel[2]. Všetky členy evolučnej rovnice levelsetu prevedieme na jednu stranu, rovnicu vynásobíme testovacou funkciou priestorovej premennej a výraz integrujeme cez oblasť riešenia Ω. Hodnota integrálu, ak ϕ je riešenie rovnice, je nulová pre všetky testovacie funkcie $\varphi \in \mathrm{L}^2(\Omega)$ a s.v. časy $t > 0$:

$$\int_\Omega \left(\frac{\partial \phi}{\partial t} + \vec{\nabla}\phi \cdot \vec{V} + F_{\max}|\vec{\nabla}\phi| \right) \varphi(\vec{x}) \mathrm{d}\vec{x} = 0 \qquad \forall\ \varphi \in \mathrm{L}^2(\Omega), \text{ s.v. } t > 0.$$

Diskretizácia

Oblasť riešenia bude v tejto práci vždy štvorec alebo obdĺžnik. Priestorové súradnice budeme značiť $\vec{x} = (x,y)$. Oblasť diskretizujeme v priestore rovnomerným delením s rovnakým priestorovým krokom h v oboch smeroch. Elementy siete sú teda vždy štvorce. Typicky v tejto práci je oblasť delená v každom smere na rádovo 100 elementov. Na sieti budeme levelset aproximovať po častiach bilineárne. Na každom elemente e je levelset aproximovaný funkciou $\hat{\phi}|_e = a_0^e + a_1^e x + a_2^e y + a_3^e xy$.

[1] Keďže je to nelineárna PDR prvého rádu, znamená to len vynásobenie rovnice testovacou funkciou a integrovanie cez oblasť riešenia bez prenášania derivácií na testovaciu funkciu.

[2] Cieľom práce nie je študovať analytické vlastnosti evolučnej rovnice.

Používame adaptívny časový krok. Veľkosť časového kroku τ_n v každom kroku[3] volíme tak, aby sa vzducholoď za časový krok nemohla posunúť o viac ako priestorový krok delený faktorom $\mu \geq 1$:

$$(\max_{\Omega} |\vec{V}(\vec{x},t_n)| + F_{\max})\tau_n = \frac{h}{\mu}.$$

Volíme implicitnú časovú schému Crankovu-Nicolsonovej, ktorá je druhého rádu presnosti v čase. V každom časovom kroku n riešime rovnicu v slabej formulácii

$$\int_\Omega \Big(\frac{\hat{\phi}^{n+1} - \hat{\phi}^n}{\tau_n} + \frac{1}{2}(\vec{\nabla}\hat{\phi}^{n+1} \cdot \vec{V}^{n+1} + F_{\max}|\vec{\nabla}\hat{\phi}^{n+1}| + \vec{\nabla}\hat{\phi}^n \cdot \vec{V}^n + F_{\max}|\vec{\nabla}\hat{\phi}^n|) \Big) \varphi \mathrm{d}\vec{x} = 0.$$

Riešenie rovnice v slabej formulácii

Rovnica v slabej formulácii vyššie je v metóde konečných prvkov reprezentovaná sústavou nelineárnych rovníc, ktorých je toľko, koľko je uzlov siete[4]. Jej riešením sú koeficienty aproximácie $\hat{\phi}^{n+1}$ na sieti v nasledovnom časovom kroku. Označme vektor týchto koeficientov (zoradených vo vhodnom poradí) $\vec{a}$. Nelineárnu sústavu, označme ju $\vec{T}(\vec{a}) = \vec{0}$, riešime Newtonovou metódou (hodnota $\vec{T}(\vec{a})$ je vektor, ktorého zložky predstavujú hodnoty integrálu pre jednotlivé numerické testovacie funkcie). Počiatočná aproximácia $\vec{a}_0$ je daná aproximáciou riešenia v aktuálnom časovom kroku $\hat{\phi}^n$. V každej iterácii Newtonovej metódy sa nasledovná aproximácia riešenia $\vec{a}_{k+1}$ spočíta ako:

$$\vec{a}_{k+1} = \vec{a}_k - (\vec{\nabla}\vec{T}(\vec{a}_k))^{-1}\vec{T}(\vec{a}_k).$$

Pre spočítanie nasledovnej aproximácie riešenia sústavy je potrebné vyriešiť sústavu lineárnych rovníc (rovnakého počtu, ako je nelineárnych rovníc) príslušnú výrazu $(\vec{\nabla}\vec{T}(\vec{a}_k))^{-1}\vec{T}(\vec{a}_k)$. Túto sústavu riešime LU rozkladom pomocou priameho riešiča MUMPS. Newtonova metóda prebieha dovtedy, dokým nie je splnené niektoré zo zastavovacích kritérií. Používame tri zastavovacie kritériá zároveň: absolútna tolerancia (2-norma rezidua menšia ako 10^{-8}), relatívna tolerancia (2-norma rezidua voči $||\vec{T}(\vec{a}_0)||_2$, teda počiatočnému odhadu[5], menšia ako 10^{-10}) a maximálny počet iterácií 20. Ak sa iterácia zastavila kvôli prvému alebo druhému kritériu, považujeme poslednú aproximáciu za riešenie, v opačnom prípade prehlásime zlyhanie riešiča. Hlavným kritériom je pre túto prácu absolútna tolerancia a relatívna tolerancia slúži ako poistka pre prípad, že by reziduum v počiatočnom odhade bolo príliš veľké.

Zastavovacie kritériá prvej fázy

Zastavovacím kritériom pre prvú fázu je $\hat{\phi}(\vec{x}_f,t) < 0$, teda dosiahnutie cieľa frontom dosiahnuteľnosti, alebo $t > T_{max}$, teda uplynutie stanoveného maximálneho času riešenia. Ak sa riešič zastaví kvôli prvému kritériu, riešenie je úspešné,

[3]Teda $t_{n+1} = t_n + \tau_n$.

[4]Pretože riešime jednu skalárnu parciálnu diferenciálnu rovnicu a používame po častiach bilineárnu aproximáciu.

[5]`https://petsc.org/release/manual/snes/#convergence-tests`

aktuálny čas t_N uložíme ako $\hat{T}_f$ a pokračujeme druhou fázou. Ak dosiahneme čas $T_{\max}$ a front nepretol cieľ, prehlásime, že riešenie neexistuje. Druhé kritérium sa dá interpretovať ako presiahnutie maximálneho prípustného času letu zadaného používateľom. Čas letu nie je ohraničený (uvažujme konštantný protivietor o rýchlosti $F_{\max} - \varepsilon$, kde $\varepsilon > 0$ je malé) alebo dokonca riešenie nemusí existovať (protivietor rýchlejší ako vzducholoď), čo sa numericky prejaví ako nekonečný čas letu. Preto je druhé kritérium potrebné.

Poznámky

Okrem levelsetu v každom časovom kroku v implementácii prvej fázy ukladáme aj veľkosť adaptívneho časového kroku τ_n a čas t_n v danom kroku (t.j. kumulatívny súčet predošlých časových krokov). Tieto hodnoty sú potrebné pre riešenie druhej fázy, vykresľovanie výstupov atď.

Maximálnu rýchlosť vetra v danom čase t_n pre určenie adaptívneho časového kroku počítame ako maximum z veľkosti $\vec{V}(\vec{x},t_n)$ cez všetky uzlové body siete.

3.2 Reinicializácia

Riešič popísaný v časti 3.1 nie je použiteľný pre väčšinu netriviálnych zadaní. Numerické chyby a veterné pole (okrem triviálnych prípadov) spôsobujú deformáciu levelsetu mimo front, takže už nepredstavuje znamienkovú vzdialenosť. Zároveň hodnoty levelsetu mimo front nie sú pre úlohu podstatné. Táto deformácia spôsobuje vážne numerické komplikácie a riešič kvôli nim zlyháva už po niekoľkých prvých krokoch. Preto je potrebné umelo zasiahnuť do levelsetu počas riešenia prvej fázy tak, aby ϕ predstavovalo znamienkovú vzdialenosť od frontu – reinicializovať. Reinicializujeme po každom časovom kroku. Pre levelset ϕ definujeme znamienkovú vzdialenosť od frontu ako:

$$d(\vec{x}) := \operatorname{sign}\phi(\vec{x}).\min_{\vec{y}\in\Gamma_0} ||\vec{x} - \vec{y}||_2.$$

Reinicializačný algoritmus dostane na vstupe levelset $\hat{\phi}'^{n+1}$ (v tejto časti budeme značiť skrátene $\hat{\phi}'$) po spočítaní jedného časového kroku. Predpokladáme, že mimo front nie je $\hat{\phi}'$ znamienková vzdialenosť. Na výstupe algoritmus vráti opravený levelset $\hat{\phi}$, ktorý má rovnakú nulovú úrovňovú množinu ako $\hat{\phi}'$ a spĺňa podmienku znamienkovej vzdialenosti. Hodnotu levelsetu v spočítanom kroku nastavíme na $\hat{\phi}^{n+1} := \hat{\phi}$ a tento levelset použijeme pre výpočet ďalšieho kroku. Na reinicializáciu použijeme fast marching method (FMM) tak, ako je popísaná v knihe Level Set Methods and Fast Marching Methods [4], s. 86–100.

Statická Hamilton-Jacobiho rovnica

Znamienková vzdialenosť spĺňa statickú Hamilton-Jacobiho rovnicu

$$|\vec{\nabla} d(\vec{x})| = 1 \quad \forall \vec{x} \in \Omega$$

s podmienkou $\operatorname{sign} d(\vec{x}) = \operatorname{sign} \hat{\phi}'(\vec{x})$ na Ω, špeciálne $d(\vec{x}) = 0$ na Γ_0. Ponúka sa možnosť reinicializovať priamo riešením tejto rovnice napr. metódou konečných

prvkov. Implementovali sme reinicializáciu riešením rovnice

$$\frac{\partial \tilde{\phi}}{\partial s}(\vec{x},s) + \operatorname{sign} \hat{\phi}'(\vec{x},t) \cdot (|\vec{\nabla}\tilde{\phi}(\vec{x},s)| - 1) = 0$$

pre $\tilde{\phi}(\vec{x},s)$ s počiatočnou podmienkou $\tilde{\phi}(\vec{x},0) = \hat{\phi}'(\vec{x},t)$. Presné riešenie tejto rovnice konverguje pre $s \to \infty$ ku znamienkovej vzdialenosti, pričom zachováva nulovú úrovňovú množinu. Tento prístup sa ukázal ako numericky nespoľahlivý a časová náročnosť výpočtu bola príliš veľká. FMM rieši statickú Hamilton-Jacobiho rovnicu, ale keďže náš prístup k reinicializačnému algoritmu je čisto informatický, tento fakt budeme využívať skôr pre názornosť ako pre výpočty.

3.3 Fast marching method

Fast marching method (FMM) je založená na princípe Djikstrovho algoritmu, grafového algoritmu pre hľadanie vzdialenosti všetkých vrcholov od zadaného vrcholu v ohodnotenom neorientovanom grafe. Implementuje sa ako grafový algoritmus a preto je potrebné levelset $\hat{\phi}'$ na vstupe previesť na graf. Levelset budeme reprezentovať grafom, ktorého vrcholy predstavujú uzly siete a hrany spájajú dvojice uzlov, ktoré sa líšia v práve jednej priestorovej súradnici o h (a druhú majú rovnakú). Hodnoty vo vrcholoch sú hodnoty levelsetu v uzlových bodoch siete a všetky hrany majú hodnotu h, teda dĺžka hrany je veľkosť priestorového kroku[6].

Označme pre potreby tejto časti $\vec{x}_{ij}$ súradnice uzla siete priradenému k vrcholu grafu v i-tom riadku a j-tom stĺpci grafu. Ďalej označme $\phi_{ij} := \hat{\phi}(\vec{x}_{ij})$.

Djikstrov algoritmus

Keďže FMM je modifikáciou Djikstrovho algoritmu, najprv popíšeme Djikstrov algoritmus a potom na jeho princípoch postavíme FMM. Djikstrov algoritmus má na vstupe ohodnotený neorientovaný graf a jeden východzí vrchol. V našom prípade je graf štvorcová sieť. Algoritmus spočíta pre každý vrchol jeho vzdialenosť od východzieho vrcholu, ktorá je daná ako najkratšia cesta do tohto vrcholu z východzieho vrcholu (dĺžkou cesty rozumieme súčet hodnôt hrán tejto cesty). Odtiaľ je zrejmé, že Djikstrov algoritmus v našom prípade rieši reinicializáciu v 1-norme.

Algoritmus postupne spracováva vrcholy. Vrcholy grafu sú rozdelené do troch kategórií (stavov): známe (2), otvorené (1) a ďaleké (0). Pre stručnosť budeme používať obrat „vrchol je v stave (1)“, resp. „vrchol je (1)“ namiesto „vrchol je otvorený“. Veľkosťou vrcholu budeme rozumieť hodnotu v tomto vrchole. Otvorené vrcholy ukladáme v halde, aby sme mohli vyberať najmenší z nich a pridávať nový vrchol do (1) v logaritmickom čase. Na začiatku je východzí vrchol v stave (2)[7] a

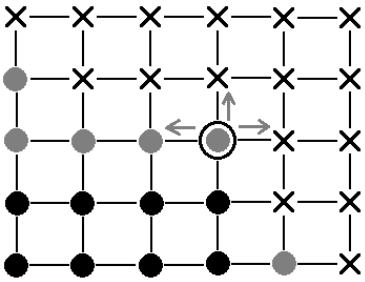
[6] Z tohto dôvodu používame všade ako elementy siete štvorce.

[7] V kontexte FMM dáva zmysel východzí vrchol označiť (2) už pri inicializácii algoritmu, narozdiel od bežnejšieho prístupu, kedy sa východzí vrchol označí (1) a na (2) sa preznačí až po prvom kroku algoritmu (teda vrchol je (1) práve vtedy, ak je v halde). Keďže v tomto prípade je východzí vrchol jeden, na jeho označení nezáleží, rozhodujúci je fakt, že sa na začiatku nachádza ako jediný v halde.

všetky ostatné vrcholy sú (0). V halde je východzí vrchol. Hodnota vo východzom vrchole je 0 a v ostatných vrcholoch je nedefinovaná, resp. $+\infty$.

Beh algoritmu (znázornený na obr. 3.1) je nasledovný:

```
dokým je halda neprázdna:
    vyber najmenší vrchol z haldy v
    prepíš stav v na (2)
    pre každého suseda u vrcholu v:
        ak u je (2): nič
        ak u je (1): prepočítaj hodnotu u podľa v
        ak u je (0):
            prepočítaj hodnotu u podľa v
            pridaj u do haldy
            prepíš stav u na (1)
```



Obr. 3.1 Priebeh Djikstrovho algoritmu. Krížiky sú vrcholy v stave (0), sivé vrcholy sú (1) a čierne sú (2). Všetky vrcholy v stave (1) sú v halde, algoritmus z nich vyberie najmenší, označí ho (2) a podľa neho prepočíta všetkých jeho susedov, ktorí nie sú (2).

Prepočet hodnoty vrcholu `u` podľa `v` prebieha tak, že do vrcholu `v` sa napíše minimum z hodnoty `u` a hodnoty `v` zväčšenej o hodnotu hrany spájajúcej `u` a `v`. V našom prípade štvorcovej siete, ak prepočítavame ϕ_{ij} podľa ϕ_{IJ}, nastavíme $\phi_{ij} := \min\{\phi_{ij}, \phi_{IJ} + h\}$.

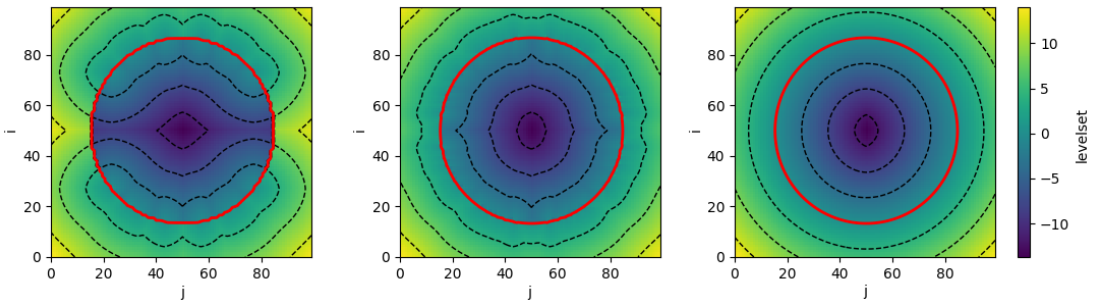
Ak je graf súvislý (v našom prípade zrejme je), po skončení algoritmu každý vrchol má spočítanú hodnotu a je v stave (2). Algoritmus beží v čase $\mathcal{O}(N \log N)$, kde N je počet vrcholov grafu (v našom prípade je počet hrán grafu $\mathcal{O}(N)$).

Inicializácia FMM

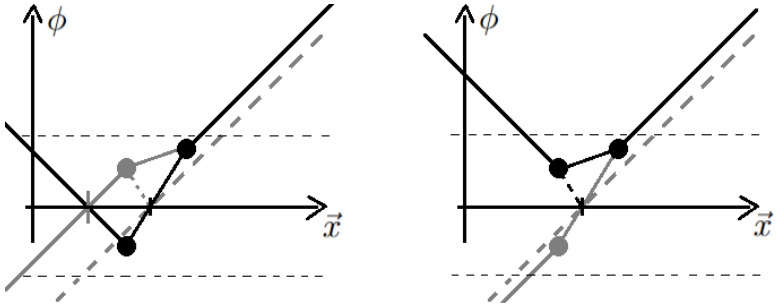
Kľúčové pre FMM je správne identifikovať front levelsetu $\hat{\phi}'$ na vstupe. To znamená určiť, ktoré vrcholy budeme považovať za východzie pre FMM.

Za východzie vrcholy, teda vrcholy na fronte (presnejšie tesne pri fronte), prehlásime všetky vrcholy s hodnotou menšou ako tolerancia v absolútnej hodnote. Týmto vrcholom nebudeme meniť hodnotu. Ako toleranciu volíme priestorový krok h. Keby bola tolerancia príliš malá, vo fronte, ktorý predstavuje spojitú krivku, by umelo vznikali diery a to by sa prejavilo ako nepresnosti vo výsledku, pri väčšom rozsahu dier dokonca ako nespojitosť výstupného levelsetu na fronte. Pri príliš veľkej tolerancii by sme ponechali viac vrcholov nezmenených, ako je nutné, a teda by nám ostala väčšia časť levelsetu neopravená (práve v okolí frontu, kde je presnosť najviac podstatná). Problematiku ilustruje obr. 3.2. Keďže v predošlom kroku bol levelset znamienková vzdialenosť, je rozumné sa spoliehať na to, že sa

za jeden krok riešiča od znamienkovej vzdialenosti veľmi neodchýlil. V prípade, že levelset je presne znamienková vzdialenosť, vrchol je v susedstve frontu ak je jeho hodnota menšia ako h v absolútnej hodnote. Keďže nevieme, či po jednom kroku je $|\vec{\nabla}\phi| > 1$ alebo $|\vec{\nabla}\phi| < 1$, voľba tolerancie ako h je rozumná.



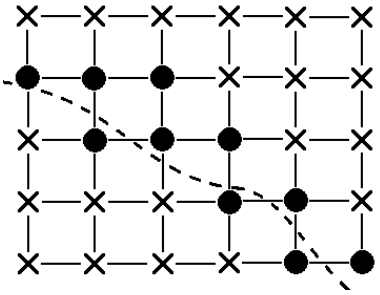
Obr. 3.2 Tri rôzne voľby tolerancie pri identifikácii frontu vo FMM, pre názornosť ilustrované na extrémnych príkladoch deformácie vstupného levelsetu, aké v praxi nestretneme. Súradnice predstavujú indexy uzlov grafu v mriežke. Vľavo: príliš malá tolerancia, vo fronte vznikajú diery a výstupný levelset je nespojitý. Stred: príliš veľká tolerancia, okolo frontu zbytočne ostal deformovaný levelset. Vpravo: vhodná voľba tolerancie.



Obr. 3.3 Identifikácia frontu vo FMM. Prerušovaná čiara znázorňuje presné riešenie reinicializácie. Zvýraznené sú uzly identifikované ako blízke frontu – tým sa nezmení hodnota. Čierna čiara znázorňuje výstup z prechádzania vrcholov, sivou je znázornený výstup z FMM po prenásobení znamienkom. Horizontálne línie sú hranice $|\phi| < h$. Vľavo: na začiatku sme nechali hodnoty pri fronte aj so znamienkom a to spôsobilo posun frontu a systematickú chybu. Vpravo: inicializácia FMM absolútnou hodnotou ϕ, chyba sa redukovala na deformáciu levelsetu v blízkosti frontu.

Všetky vrcholy spĺňajúce $|\phi(\vec{x}_{ij})| < h$ označíme (2) a pridáme do haldy[8]. Ich hodnotu nastavíme na $\phi_{ij} := |\phi(\vec{x}_{ij})|$. Keďže FMM tak ako Djikstrov algoritmus počíta absolútnu vzdialenosť (bez znamienka), ak by sme ponechali v stave (2) záporné hodnoty, vnútri frontu by sme tým vyrobili systematickú chybu a následne falošný front tesne vedľa správneho. Viď obr. 3.3. Všetky ostatné vrcholy nech majú hodnotu nedefinovanú (resp. nekonečnú). Viď obr. 3.4. Následne prebieha spracovanie vrcholov spôsobom Djikstrovho algoritmu.

[8]Tu sa nemôžeme držať konvencie, že vrchol je (1) práve ak je v halde, pretože potrebujeme zabezpečiť, že východzie hodnoty na fronte sa nezmenia.



Obr. 3.4 FMM po identifikácii frontu. Čierne vrcholy sú vrcholy na fronte, ktorým sa nezmení hodnota, tie sú v stave (2) a všetky sú v halde. Krížiky sú vrcholy v stave (0) s nedefinovanou hodnotou. Na obrázku je veľkosť gradientu deformovaného levelsetu v hornej časti menšia ako 1 a v dolnej časti je väčšia ako 1.

Spracovanie vrcholov

FMM prechádza vrcholy rovnakým spôsobom ako Djikstrov algoritmus. Líši sa len v prepočte suseda spracovávaného vrcholu. Kým Djikstrov algoritmus prepočíta hodnotu v každom susede ϕ_{ij} vrcholu ϕ_{IJ}, ak ϕ_{ij} nie je (2), vzťahom

$$\phi_{ij} := \min\{\phi_{ij}, \phi_{IJ} + h\},$$

FMM prepočíta hodnotu ϕ_{ij} podľa všetkých jeho susedov tak, aby bola splnená rovnica

$$1 = (\max\{D_{ij}^{-x}\phi, -D_{ij}^{+x}\phi, 0\})^2 + (\max\{D_{ij}^{-y}\phi, -D_{ij}^{+y}\phi, 0\})^2, \tag{3.1}$$

kde $D_{ij}^{+x}\phi = \frac{\phi_{(i+1)j} - \phi_{ij}}{h}$, a D_{ij}^{-x},D_{ij}^{+y},D_{ij}^{-y} je definované analogicky.

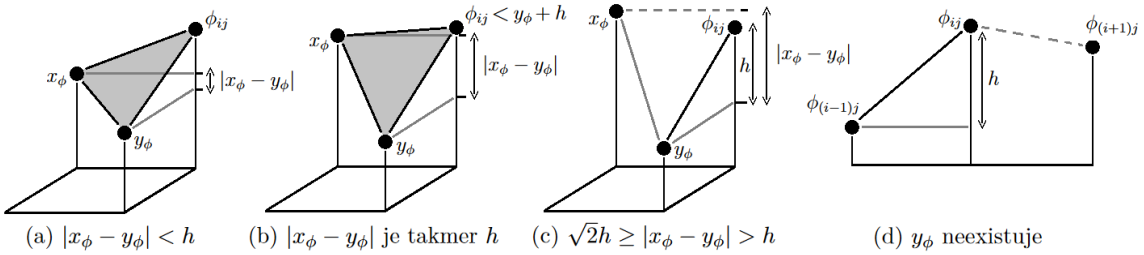
Túto rovnicu sme museli vyriešiť, aby sme získali inštrukcie pre prepočet vrcholu ([4] inštrukcie riešiace rovnicu neuvádza). Jej riešenie je z priestorových dôvodov popísané v časti 3.4 a tu uvedieme len výsledný algoritmus. Pre stručnosť budeme hovoriť, že vrchol existuje, ak je to vrchol grafu, ktorý nie je (0). Pre každého suseda ϕ_{ij} práve spracovávaného vrcholu ϕ_{IJ} vykonáme nasledovný prepočet v prípade, že ϕ_{ij} nie je (2):

- ak existuje $\phi_{(i+1)j}$ aj $\phi_{(i-1)j}$, definujme $x_\phi := \min\{\phi_{(i+1)j}, \phi_{(i-1)j}\}$,
- ak existuje len jedno z nich, definujeme ním x_ϕ,
- inak nech x_ϕ neexistuje,
- ten istý postup vykonáme pre smer y,
- ak existuje x_ϕ aj y_ϕ:
  - ak $|x_\phi - y_\phi| < h$, položíme $\phi_{ij} = \frac{1}{2}(x_\phi + y_\phi + \sqrt{2h^2 - (x_\phi - y_\phi)^2})$,
  - inak[9] $\phi_{ij} = \min\{x_\phi, y_\phi\} + h$,
- ak existuje len jedno z x_ϕ, y_ϕ, definujeme ním z_ϕ a položíme $\phi_{ij} = z_\phi + h$, tým sme vyčerpali všetky možnosti[10].

[9] Ako sa ukáže v časti 3.4, vždy je $|x_\phi - y_\phi| \leq \sqrt{2}h$, ak by sa počas FMM ukázalo, že to neplatí, je to spôsobené konečnou aritmetikou. V kóde sa táto nerovnosť nekontroluje.

[10] Vždy existuje aspoň jedno z x_ϕ, y_ϕ (uvidíme v časti 3.4). Ak by neexistovalo ani jedno, jedná sa o chybu v kóde a program v súčasnej implementácii spadne. To sa nedeje.

Prepočet je graficky znázornený na obrázku 3.5. Týmto spôsobom spočíta algoritmus absolútnu vzdialenosť každého vrcholu od frontu.



(a) $|x_\phi - y_\phi| < h$ (b) $|x_\phi - y_\phi|$ je takmer h (c) $\sqrt{2}h \geq |x_\phi - y_\phi| > h$ (d) y_ϕ neexistuje

Obr. 3.5 Prepočet vrcholu ϕ_{ij} vo FMM. V prípadoch (a), (b), (c) existuje x_ϕ aj y_ϕ, v prípade (d) neexistuje y_ϕ a je znázornená voľba x_ϕ.

Prevod na znamienkovú vzdialenosť

Po skončení hlavnej časti programu sa hodnota v každom vrchole prenásobí znamienkom levelsetu na vstupe, čím sa splní znamienková konvencia a požiadavka na znamienkovú vzdialenosť. Výsledná funkcia sa ako hodnoty v uzlových bodoch vracia na výstup.

Poznámky

FMM, ako aj Djikstrov algoritmus, sme naprogramovali v čistom Pythone bez použitia akýchkoľvek knižníc. Tak isto sme implementovali haldu pre otvorené vrcholy. Naprogramovali sme aj verziu používajúcu na uloženie grafu `numpy.array` namiesto Pythonovského 2D poľa, táto verzia však bola dvojnásobne pomalšia ako verzia v čistom Pythone.

Reinicializácia trvá oproti riešeniu problému v slabej formulácii Newtonovou metódou zanedbateľný čas.

3.4 Riešenie rovnice prepočtu vrcholu vo FMM

V tejto časti popíšeme postup, akým sme z rovnice (3.1) odvodili inštrukcie pre prepočet vrcholu ϕ_{ij}. Riešime rovnicu

$$1 = (\max\{D_{ij}^{-x}\phi, -D_{ij}^{+x}\phi, 0\})^2 + (\max\{D_{ij}^{-y}\phi, -D_{ij}^{+y}\phi, 0\})^2$$

pre vrchol ϕ_{ij}, ak tento vrchol nie je (2), pričom hodnoty všetkých jeho susedov považujeme za dané a v rovnici je jedna neznáma - hodnota ϕ_{ij}. Začneme pozorovaním:

$$D_{ij}^{+x}\phi = \frac{\phi_{(i+1)j} - \phi_{ij}}{h} < 0 \iff \phi_{ij} > \phi_{(i+1)j}.$$

Podobne

$$D_{ij}^{-x}\phi = \frac{\phi_{ij} - \phi_{(i-1)j}}{h} > 0 \iff \phi_{ij} > \phi_{(i-1)j}.$$

Rovnaký vzťah platí pre smer y.

Zrejme nezahŕňame do výpočtu susedné vrcholy v stave (0), čo plynie z toho, že ich hodnota je nedefinovaná (resp. $+\infty$). Podobne ako v predošlej časti budeme

hovoriť, že vrchol neexistuje, ak to nie je vrchol grafu alebo je (0). V ďalších výpočtoch nebudeme BUNV neexistenciu vrcholu uvažovať a budeme predpokladať, že neexistujúce vrcholy majú hodnotu $+\infty$, čo nemá na výsledky výpočtov vplyv[11].

Vidíme, že vo výpočte maxima v rovnici (3.1) môžu nastať tri možnosti:

- obe susedné hodnoty neexistujú $\implies$ príslušný člen je nulový,
- existuje jedna susedná hodnota $\implies$ príslušný člen je nulový alebo je ňou určený,
- existujú obe susedné hodnoty $\implies$ príslušný člen je určený menšou[12] z nich alebo nulový.

Všimnime si, že vždy existuje aspoň jedna susedná hodnota - vrchol ϕ_{IJ}, ktorý bol práve preznačený z (1) na (2). To znamená, že vždy je nejaký člen na pravej strane rovnice (3.1) nenulový. Zároveň ak je daný člen nenulový, je jeho hodnota určená menším zo susedov (resp. jediným, ak druhý sused neexistuje). Preto môžeme BUNV uvažovať len $D_{ij}^{+x}\phi$ a $D_{ij}^{+y}\phi$. Tento predpoklad, ktorý je BUNV, sa premietne do výsledného postupu ako voľba x_ϕ resp. y_ϕ.

Ak je nenulový len jeden člen, BUNV nech je to x_ϕ (prejaví sa ako voľba z_ϕ). Z úvah vyššie plynie, že $y_\phi \geq \phi_{ij} > x_\phi$ a preto je $y_\phi > x_\phi$ (ak oba existujú).

Doterajšie úvahy môžeme zhrnúť nasledovne. V rovnici (3.1) nastane práve jeden z prípadov:

- oba členy sú nenulové, predpoklad $\phi_{ij} > x_\phi$ a $\phi_{ij} > y_\phi$,
- len člen s menším susedným ϕ je nenulový.

Budeme riešiť rovnicu (3.1) pre dva rôzne prípady a následne overíme predpoklady. Dostaneme riešenie ϕ_{ij} pre rôzne prípady.

Riešenie pre oba členy nenulové

BUNV uvažujeme len $D_{ij}^{+x}\phi$ a $D_{ij}^{+y}\phi$ a označme $x_\phi := \phi_{(i+1)j}$ a $y_\phi := \phi_{i(j+1)}$. Predpokladáme, že oba členy v rovnici (3.1) sú nenulové, čo znamená, že $\phi_{ij} > x_\phi$ a $\phi_{ij} > y_\phi$. Rovnica má za týchto predpokladov tvar

$$1 = \left(-\frac{x_\phi - \phi_{ij}}{h}\right)^2 + \left(-\frac{y_\phi - \phi_{ij}}{h}\right)^2,$$

čo upravíme do tvaru:

$$0 = \phi_{ij}^2 - \phi_{ij}(x_\phi + y_\phi) + \frac{1}{2}(x_\phi^2 + y_\phi^2 - h^2).$$

Diskriminant tejto rovnice je $2h^2 - (x_\phi - y_\phi)^2$ a teda pre existenciu riešenia potrebujeme $|x_\phi - y_\phi| \leq \sqrt{2}h$. Táto podmienka má jasný geometrický význam, pretože vzdialenosť medzi vrcholmi x_ϕ a y_ϕ je $\sqrt{2}h$ a $|\vec{\nabla}\phi| = 1$.

[11] V implementácii majú neexistujúce vrcholy hodnotu nedefinovanú.

[12] Teda hodnotou s väčším rozdielom v absolútnej hodnote oproti hodnote vrcholu ϕ_{ij}, ktorú ale nepoznáme.

Riešenie kvadratickej rovnice je

$$\phi_{ij} = \frac{1}{2}\left(x_\phi + y_\phi \pm \sqrt{2h^2 - (x_\phi - y_\phi)^2}\right).$$

Overíme predpoklady.

$$\begin{aligned}
\phi_{ij} > x_\phi \quad &\text{práve vtedy, ak} \quad -(x_\phi - y_\phi) \pm \sqrt{2h^2 - (x_\phi - y_\phi)^2} > 0, \\
\phi_{ij} > y_\phi \quad &\text{práve vtedy, ak} \quad (x_\phi - y_\phi) \pm \sqrt{2h^2 - (x_\phi - y_\phi)^2} > 0.
\end{aligned}$$

Okamžite vidieť, že aby boli splnené obe nerovnosti zároveň, potrebujeme vybrať riešenie s kladným znamienkom odmocniny a musí platiť

$$|x_\phi - y_\phi| < \sqrt{2h^2 - (x_\phi - y_\phi)^2}.$$

Vieme, že $|x_\phi - y_\phi| \leq \sqrt{2}h$ a teda môžeme nerovnosť upraviť ekvivalentne na $|x_\phi - y_\phi| < h$.

Dokopy dostávame výsledok:

$$\phi_{ij} = \frac{1}{2}\left(x_\phi + y_\phi + \sqrt{2h^2 - (x_\phi - y_\phi)^2}\right) \quad \text{ak} \quad |x_\phi - y_\phi| < h.$$

V opačnom prípade nemožno splniť predpoklady a musíme pristúpiť k riešeniu pre jeden nulový člen.

Riešenie pre jeden nenulový člen

BUNV uvažujeme len $D_{ij}^{+x}\phi$ a $D_{ij}^{+y}\phi$ a pri doterajšom označení naviac $x_\phi < y_\phi$. Predpokladáme člen príslušný smeru x nenulový a člen pri y nulový, to znamená $y_\phi \geq \phi_{ij} > x_\phi$. Rovnica má tvar

$$1 = \left(-\frac{x_\phi - \phi_{ij}}{h}\right)^2,$$

čo upravíme do tvaru:

$$0 = \phi_{ij}^2 - 2x_\phi\phi_{ij} + (x_\phi^2 - h^2).$$

Rovnica má diskriminant $4h^2$, ktorý je vždy kladný a teda riešenie vždy existuje. Riešenie rovnice je

$$\phi_{ij} = x_\phi \pm h.$$

Z predpokladu $\phi_{ij} > x_\phi$ plynie, že volíme znamienko plus. Overíme predpoklad $y_\phi \geq \phi_{ij}$ (ak y_ϕ existuje). Dostávame $y_\phi - x_\phi \geq h$. Všimnime si, že $|y_\phi - x_\phi| \geq h$ je doplnkové s predpokladom pre riešenie rovnice s oboma členmi nenulovými. Zároveň z predpokladov BUNV je $x_\phi \leq y_\phi$ a teda je nerovnosť splnená aj bez absolútnej hodnoty. Celkovo sme ukázali, že rovnica (3.1) má vždy jednoznačné riešenie. V prípade, že je jeden člen nulový a jeden nenulový, máme riešenie:

$$\phi_{ij} = x_\phi + h \quad \text{ak} \quad |x_\phi - y_\phi| \geq h.$$

Môžeme si všimnúť, že ak $|x_\phi - y_\phi| = h$, dávajú oba vzťahy pre riešenie ten istý výsledok (máme BUNV $x_\phi < y_\phi$ a teda $y_\phi = x_\phi + h$).

3.5 Druhá fáza výpočtu

Ak prvá fáza skončila úspešne, teda prekročením cieľa frontom, pokračuje výpočet druhou fázou. Z prvej fázy poznáme čas cesty $\hat{T}_f$ a levelset v každom časovom kroku (a tiež veľkosti časových krokov v každom kroku). Ďalej z teórie vieme, že vzducholoď sa v čase T_f, kedy front pretne cieľ, nachádza v cieľovom bode $\vec{x}_f$ a že jej trasu môžeme rekonštruovať riešením rovnice (2.6):

$$\frac{\mathrm{d}\vec{p}}{\mathrm{d}t} = \vec{V}(\vec{p},t) + F_{\max}\frac{\vec{\nabla}\phi(\vec{p},t)}{|\vec{\nabla}\phi(\vec{p},t)|}$$

spätne v čase s počiatočnou podmienkou $\vec{p}(T_f) = \vec{x}_f$ do času $t = 0$ (resp. do času štartu, ak je nenulový). Výraz $\frac{\vec{\nabla}\phi}{|\vec{\nabla}\phi|}$ predstavuje natočenie kormidla v danom čase.

Počas druhej fázy teda ukladáme natočenie kormidla a trajektóriu. Zrejme front posledného levelsetu neobsahuje cieľový bod, ale front ho mierne (nanajvýš o $\frac{h}{\mu}$) preskočil. Keďže potrebujeme počítať výstupy z bodov na fronte, najprv projektujeme $\vec{x}_f$ na posledný levelset a integráciu rovnice (2.6) začíname v projektovanom bode, teda numerická počiatočná podmienka je $\vec{p}_N = \vec{x}_f|_{\hat{\phi}^N}$. Projekciu počítame ako $\vec{x}_f|_{\hat{\phi}^N} = \vec{x}_f - \vec{\nabla}\hat{\phi}^N(\vec{x}_f)\frac{\hat{\phi}^N(\vec{x}_f)}{|\vec{\nabla}\hat{\phi}^N(\vec{x}_f)|^2}$.

Následne integrujeme rovnicu explicitnou Eulerovou metódou, ktorá je prvého rádu:

$$\vec{p}_{n-1} = \vec{p}_n - \tau_{n-1}\left(\vec{V}(\vec{p}_n,t_n) + F_{\max}\frac{\vec{\nabla}\hat{\phi}^n(\vec{p}_n)}{|\vec{\nabla}\hat{\phi}^n(\vec{p}_n)|}\right).$$

V každom kroku sa posunie vzducholoď o jeden levelset dozadu. Kormidlo pre daný krok sa počíta z aktuálneho levelsetu. Používame vietor z aktuálneho kroku n a časový krok z predošlého kroku $n-1$ (ktorý sa počítal na základe vetra v čase t_{n-1}), pretože na základe τ_{n-1} sa počítal posun frontu do kroku n. Preto môže byť občas CFL podmienka mierne[13] porušená. Ukladáme kormidlo $\vec{h}_n = \frac{\vec{\nabla}\hat{\phi}^n(\vec{p}_n)}{|\vec{\nabla}\hat{\phi}^n(\vec{p}_n)|}$ a trasu $\vec{p}_n$. Počiatočný levelset daný ako $\phi(\vec{x},0) = |\vec{x} - \vec{x}_s|$ nemá normálu k frontu a v tomto kroku presné riešenie dosiahlo štartovný bod. Preto v čase t_0 už nepočítame kormidlo $\vec{h}_0$ a neaktualizujeme polohu vzducholode[14]. Keď dosiahneme tento krok, končí výpočet druhej fázy.

Keďže levelset predstavuje znamienkovú vzdialenosť, ak by sme mali z prvej fázy presné levelsety vo všetkých krokoch, mohli by sme pomocou hodnoty $\hat{\phi}^n(\vec{p}_n)$, ktorá je pri presnom výpočte nulová, určovať priebežnú chybu. Avšak ako sme videli v časti o FMM, hodnoty levelsetu v blízkosti frontu sa neupravujú a preto má tento odhad chyby obmedzenú výpovednú hodnotu. Z toho istého dôvodu v numerickej spätnej integrácii ponechávame normovanie gradientu levelsetu.

Výpočet druhej fázy končíme v čase t_0 na polohe $\vec{p}_0$, ktorá je pri presnom výpočte rovná $\vec{x}_s$. Vzdialenosť $|\vec{p}_0 - \vec{x}_s|$ môžeme použiť ako ďalší odhad presnosti riešenia. Nazvime ho koncová chyba spätnej integrácie (KCS). Výpovednosť KCS

[13] Ak sa veterné pole v čase mení rozumne.

[14] Keby sme spočítali kormidlo z tohto levelsetu, numericky by sme výsledok dostali. Toto kormidlo $\vec{h}_0$ je ale čisto výsledok vlastností konečnej aritmetiky a od riešenia úlohy je pravdepodobne veľmi odchýlené. Započítanie tohto kormidla a jeho použitie v prvom kroku tretej fázy spôsobí veľké chyby, ktoré sa prejavia v KCI (viď ďalšia časť), čo sme pozorovali pri praktickej implementácii.

nie je závislá na presnosti levelsetu v okolí frontu. Jej súvislosť s riešením úlohy, teda odpoveďou na otázku ako riadiť vzducholoď, aby sa dostala (v najkratšom čase) z bodu $\vec{x}_s$ do bodu $\vec{x}_f$, je však vzdialená a preto je výpovednosť aj tohto čísla obmedzená.

3.6 Odhad chyby

V predošlej časti sme videli, že z druhej fázy výpočtu vieme síce získať všetky chýbajúce inštrukcie pre vzducholoď, ale len obmedzené informácie o ich presnosti. Preto pridávame ešte jednu fázu, ktorej myšlienka je nasledovná: ak sa bude vzducholoď riadiť spočítanými inštrukciami, ako blízko k cieľu sa dostane v čase $\hat{T}_f$?

V tretej fáze výpočtu integrujeme rovnicu pohybu vzducholode s dosadeným známym kormidlom:

$$\frac{\mathrm{d}\vec{q}}{\mathrm{d}t} = \vec{V}(\vec{q},t) + F_{\max}\vec{h}(t). \tag{3.2}$$

Počiatočná podmienka je $\vec{q}(t_0) = \vec{q}_0 = \vec{x}_s$. Opäť používame explicitnú Eulerovu metódu:

$$\vec{q}_{n+1} = \vec{q}_n + \tau_n(\vec{V}(\vec{q}_n,t_n) + F_{\max}\vec{h}_{n+1}).$$

Tretia fáza už nevyužíva spočítané levelsety, len uložené kormidlo. Avšak kormidlo máme definované len od kroku 1 do posledného kroku vrátane, v nultom kroku nemá front normálu. V poslednom kroku pre presné riešenie platí $\vec{q}(T_f) = \vec{x}_f$. Preto v poslednom kroku N už nepotrebujeme poznať kormidlo $\vec{h}_N$, ktoré by sa použilo pre výpočet polohy v ďalšom kroku, a ktoré sme v druhej fáze spočítali. Preto na výpočet polohy používame kormidlo v nasledovnom kroku $n+1$.

V presnom výpočte je $\vec{q}(T_f) = \vec{x}_f$. Vzdialenosť $|\vec{q}_N - \vec{x}_f|$ použijeme ako indikátor presnosti výpočtu a nazveme ju koncová chyba inštrukcií (KCI). Ak je presný len výpočet tretej fázy, KCI zohľadňuje nepresnosti vo výpočte levelsetu v prvej fáze, preskočenie cieľa posledným frontom (teda rozdiel $\hat{T}_f$ oproti T_f), nepresnosť integrácie v druhej fáze, aj nepresnosť určenia kormidla v druhej fáze danú deformáciou levelsetu v okolí frontu. KCI je najvýpovednejší indikátor presnosti výpočtu, ktorý používame. Jeho výpovednosť je obmedzená len presnosťou výpočtu tretej fázy. KCI ukladáme ako výstup riešiča indikujúci presnosť výpočtu.

Keďže tretia fáza je výpovednejšia, čo sa odpovede na základnú otázku týka, ukladáme trasu $\vec{q}_n$ a túto trasu prehlásime za výstup riešiča, teda vypočítanú trajektóriu vzducholode.

3.7 Záverečné poznámky

CFL parameter μ volíme aspoň 1 a spolu s počtom uzlov siete (resp. priestorovým krokom) charakterizujú jemnosť diskretizácie, pričom voľbou μ určujeme časovú diskretizáciu, ktorá je ale naviac automaticky prispôsobená konkrétnej situácii na sieti. Typicky sme volili v tejto práci $\mu = 2$.

Veľkosť časového kroku sa prepočítava v každom kroku podľa aktuálneho vetra, čo vyžaduje získať hodnoty levelsetu (uloženého ako Firedrake funkcia) vo všetkých uzloch siete a nájsť v nich maximum. Táto operácia môže trvať nezanedbateľný

čas a v konečnom dôsledku spomaľovať výpočet. To je zbytočné, ak je vietor v čase konštantný. Preto je prípad časovo premenlivého vetra implementovaný osobitne – implementácia obsahuje naviac aktualizáciu veterného poľa v každom kroku a výpočet veľkosti časového kroku.

Z teórie aj implementácie druhej fázy vyplýva, že nie je možné úlohu riešiť bez ukladania všetkých vypočítaných levelsetov, pretože druhá fáza ich používa v opačnom poradí, ako boli získané v prvej fáze. Problémy s nedostatkom pamäte sme nezaznamenali.

Počas výpočtu prvej fázy je zaujímavé sledovať, ako sa vyvíja v čase vzdialenosť frontu od cieľa. Keďže levelset po každom kroku reinicializujeme na znamienkovú vzdialenosť, stačí ho vyčísliť v bode $\vec{x}_f$. Hodnota $\phi(\vec{x}_f,t)$ nevypovedá o čase ostávajúcom do dosiahnutia cieľa (ani pri vetre, ktorý sa v čase nemení) a nemusí byť monotónnou funkciou času.

Technický detail – komunikácia Firedrake a Pythonu

Získanie hodnôt levelsetu vo všetkých uzlových bodoch je potrebné pre výpočet τ_n, ako aj pre FMM (pre ktorú naviac treba implementovať aj opačný proces). Firedrake neposkytuje priamy spôsob, ako dostať z funkcie na obdĺžnikovej sieti štvorcových elementov dvojrozmerné pole hodnôt v uzloch[15]. Je možné získať zo siete zoznam súradníc jej uzlov (v nešpecifikovanom poradí)[16]. Z funkcie sa dá získať zoznam jej hodnôt v uzloch siete[17], bez súradníc uzlov, ale v rovnakom poradí, v akom dostaneme zo siete súradnice jej uzlov. To umožňuje implementovať algoritmus, ktorý prevedie Firedrake funkciu na sieti na pole hodnôt v uzloch siete. Pre sieť $m \times n$ uzlov, zoradíme zložky všetkých súradníc v smere x (pozeráme len na prvú zložku súradníc vrcholu) vzostupne, dostaneme zoradený zoznam n rôznych hodnôt. Následne vytvoríme slovník, ktorý priradí súradnici jej index v zoradenom zozname. To isté urobíme pre smer y. Potom prechádzame súbežne zoznam súradníc sieťových uzlov a funkčných hodnôt (oba o dĺžke mn) a pomocou dvojice slovníkov priraďujeme hodnoty do príslušných pozícií v poli. Tým je zabezpečená konverzia v čase $\mathcal{O}(n\log(n) + m\log(m) + nm)$. To je lepšia časová zložitosť ako samotná FMM, ktorá je $\mathcal{O}(nm\log(nm))$, preto je tento čas zanedbateľný. Opačná konverzia funguje na rovnakom princípe. Konverzia je implementovaná ako funkcia `f_na_array` resp. `array_na_f`.

Vzducholoď opustí sieť

Môže sa stať, že vzducholoď v niektorej fáze opustí sieť. V prvej fáze to nastane, ak front celý prekročí hranicu Ω – hodnota levelsetu je na celej oblasti kladná. Riešič rovnice v slabej formulácii to dokáže spracovať. FMM v súčasnej implementácii zlyhá. Neidentifikuje front a preto začína s prázdnou haldou a všetkými vrcholmi v stave (0), teda s hodnotou nedefinovanou. Keďže halda je prázdna už na začiatku, hlavná časť programu sa nevykoná vôbec, ako keby už boli všetky vrcholy prejdené. Následne pri prenásobení hodnôt znamienkom levelsetu

[15]Je možné vyčísliť funkciu v konkrétnom bode a teda v cykle prejsť všetky uzly siete. V praxi sa to ukázalo pomalé.

[16]Pre sieť uloženú ako `mesh` je to `mesh.coordinates.dat.data`.

[17]Pre funkciu `f` je tento zoznam v premennej `f.dat.data`, zároveň sa dá premenná prepísať, čo využívame v obrátenom procese po ukončení FMM.

na vstupe Python vyhlási chybu, pretože všetky hodnoty sú nedefinované. Ak by sme aj implementáciu FMM upravili tak, aby sme nedostávali chybu, nedokázali by sme reinicializovať štandardným spôsobom a po niekoľkých krokoch by zlyhal hlavný riešič vplyvom deformácie levelsetu bez reinicializácie. Okrem pochybného praktického významu pokračovania výpočtu je toto ďalší dôvod, prečo v takom prípade zastavíme integráciu a prehlásime nenájdenie riešenia rovnako ako pri dosiahnutí času $T_{\max}$.

Ak vzducholoď neopustila sieť v prvej fáze, môže opustiť sieť v druhej fáze, ak časť frontu dosiahla počas prvej fázy hranicu oblasti. V tretej fáze je to spôsobené numerickými chybami. Firedrake dokáže vyčísliť funkciu mimo sieť do vzdialenosti približne jedného priestorového kroku, potom dostaneme chybu. V ďalších fázach kontrolujeme polohu vzducholode a ak opustí sieť (skôr ako vyhlási chybu Firedrake), zastavíme výpočet a prehlásime nenájdenie riešenia.

Spresnenie výpočtu druhej fázy

Ako počiatočnú podmienku druhej fázy používame projekciu $\vec{x}_f$ na posledný front. Alternatívnou možnosťou je obrátený prístup – interpolovať front do $\vec{x}_f$. Zistíme vzdialenosti cieľového bodu od oboch susedných frontov ako $u_1 := |\hat{\phi}_N(\vec{x}_f)|$ a $u_2 := |\hat{\phi}_{N-1}(\vec{x}_f)|$ a pomocou nich následne spočítame odhad frontu pretínajúceho $\vec{x}_f$ lineárnou interpoláciou[18]. Následne interpolovaným levelsetom predefinujeme posledný levelset a prepočítame záznamy veľkostí časových krokov τ_{N-1} (čas, ktorý ubehol medzi krokmi $N-1$ a N) a τ_N (nepoužitý, lebo výpočet skončil; prepočíta sa nanovo na základe vetra v nanovo určenom čase t_N) a času cesty $t_N = \hat{T}_f$. Používame nasledovné vzťahy:

$$\begin{aligned}
\hat{\phi}_N^{\text{interp}} &:= \hat{\phi}_N \frac{u_2}{u_1+u_2} + \hat{\phi}_{N-1} \frac{u_1}{u_1+u_2}, \\
\tau_{N-1}^{\text{interp}} &:= \tau_{N-1} \frac{u_2}{u_1+u_2}, \\
t_N^{\text{interp}} &:= t_{N-1} + \tau_{N-1}^{\text{interp}}, \\
\tau_N^{\text{interp}} &:= \frac{h}{\mu} (\max_{\Omega} |\vec{V}(\vec{x}, t_N^{\text{interp}})| + F_{\max})^{-1}.
\end{aligned}$$

Implementovali sme aj túto variantu riešiča, KCS aj KCI boli s interpoláciou porovnateľne veľké ako s projekciou. Riešenie pre problematické prúdenia sa nezlepšilo. Vysvetlenie je, že levelset ostáva v blízkosti frontu po FMM deformovaný a preto interpolačná metóda postavená na jeho presnosti zlyháva. Výsledky v tejto práci ako aj riešič pre reálne dáta používajú projekciu $\vec{x}_f$ na posledný front. Viac ku možným zlepšeniam výpočtu druhej a tretej fázy uvedieme v diskusii.

[18] u_1 a u_2 nie sú presne vzdialenosti od frontov, pretože FMM ponecháva levelset v okolí frontu deformovaný.

4 Výsledky

V tejto časti uvedieme významné výsledky, ktoré sme dosiahli použitím metód popísaných v časti 3. Najprv ukážeme zopár umelo skonštruovaných príkladov ilustrujúcich vlastnosti úlohy alebo riešiča. Následne porovnáme náš riešič s riešením úlohy s konštantným vetrom pomocou systému ODR. Nakoniec popíšeme spôsob použitia riešiča na reálne dáta a uvedieme dosiahnuté výsledky.

Ako výstup z riešiča ukladáme nasledovné údaje:

- trajektória vzducholode – textový súbor obsahujúci pre každý časový krok na samostatnom riadku tri údaje: čas a súradnice x a y vzducholode v danom čase (súradnice $\vec{q}_n$ z tretej fázy); ukladá sa vrátane bodov $\vec{x}_s$ a $\vec{q}_N$,
- navigačné inštrukcie – textový súbor obsahujúci pre každý časový krok na samostatnom riadku tri údaje: čas a smerovanie kormidla v tomto čase (ako popísané v časti 3.6 o tretej fáze) ako zložky x a y jednotkového vektora $\vec{h}_{n+1}$; ukladá sa pre $n \in \{0,1,\ldots N-1\}$,
- čas cesty $\hat{T}_f$,
- čas štartu t_0 – je významný pri použití reálnych dát, ale ukladáme ho aj v prípade, keď je nulový,
- časová pečiatka predpovede počasia – týka sa len riešiča pre reálne dáta, bližšie vysvetlenie neskôr,
- východzí bod $\vec{x}_s$,
- cieľový bod $\vec{x}_f$,
- veľkosť priestorového kroku h,
- CFL podiel μ,
- KCI.

Okrem použitia reálnych dát uvažujeme vo všetkých príkladoch bezrozmerné veličiny[1]. V riešiči pre reálne dáta meriame vzdialenosti v kilometroch, relatívny čas v hodinách od začiatku prvého dňa cesty, absolútny čas je pre túto prácu GMT. Sférické súradnice používame štandardné zemepisné súradnice.

4.1 Všeobecné postrehy

Výpočet na notebooku trvá približne do desať minút. Väčšinu času sa počíta prvá fáza. V prvej fáze trvá najdlhšie konvergencia Newtonovej metódy, resp. riešenie sústavy lineárnych rovníc. Newtonova metóda vďaka reinicializácii typicky skonverguje po 3 až 4 iteráciách. Reinicializácia trvá oproti Newtonovej metóde zanedbateľný čas. Aktualizácia veterného poľa a veľkosti časového kroku pri premenlivom vetre netrvá tak dlho ako Newtonova metóda, jej vplyv na trvanie výpočtu je malý, ale pre zlepšenie časovej efektivity je stacionárny vietor implementovaný osobitne. Výpočet druhej fázy trvá niekoľkonásobne kratšie ako výpočet prvej fázy. Tretia fáza trvá najkratšie, niekoľkokrát kratšie ako druhá

[1] Až na trvanie výpočtu, ktoré meriame v sekundách.

fáza, pretože v druhej fáze počítame a vyčísľujeme gradient levelsetu, kým v tretej fáze len čítame uložené kormidlo.

Výpočty dosahujú vzhľadom k použitým numerickým metódam veľkú presnosť. KCI je typicky menšia ako priestorový krok, rádovo aj $\frac{h}{10}$. Priebežná chyba spätnej integrácie sa typicky pohybuje zo začiatku výpočtu na hodnotách rádovo tisíciny priestorového kroku, ku koncu stúpa na rád desatín h, občas mierne prekročí h. Výpovednosť tohto čísla ale závisí na presnosti levelsetu na okolí frontu, kde FMM neopravuje deformácie. Na zlepšenie presnosti sa ponúka možnosť projektovať polohu vzducholode v každom kroku druhej fázy na aktuálny front. Keď sme implementovali túto verziu, priebežná chyba spätnej integrácie sa zvýšila o dva rády v priebehu výpočtu a na konci sa pohybovala rádovo v desatinách až jednotkách h. KCI sa niekoľkonásobne zhoršila. Vysvetľujeme si to deformáciou reinicializovaného levelsetu v okolí frontu, ktorá spôsobila znehodnocujúcu nepresnosť projekcie.

S niektorými zadaniami má úloha viacero riešení. Typický príklad je radiálne prúdenie, ktorého charakter spôsobuje, že je výhodnejšie ho obísť ako prejsť stredom. Ak sú $\vec{x}_s$ a $\vec{x}_f$ stredovo súmerné podľa stredu tohto prúdenia, úloha má dve riešenia. Matematicky sa to prejaví neexistenciou (resp. nespojitosťou) normály v niektorom bode frontu. Numericky, keď v druhej fáze vzducholoď dosiahne takýto bod, riešič typicky nezlyhá a spočítané kormidlo smeruje medzi (limitné) hodnoty normály z jednotlivých strán[2]. To vedie v prípade popísaného prúdenia k veľkým chybám výpočtu, ktoré sa prejavia (okrem iného) v KCI – čiže vieme zlyhanie tohto typu odhaliť.

4.2 Umelé príklady

V tejto časti predstavíme niekoľko príkladov riešenia úlohy vybudovaným riešičom. Príklady sú vymyslené a ich význam je demonštračný. Ak nie je povedané inak, pracujeme na štvorcovej oblasti o rozmeroch 15×15, rýchlosť vzducholode je $F_{\max} = 1$ a sieť má 100 elementov v každom smere (teda $h = 0{,}15$). Čas štartu je $t_0 = 0$. CFL podiel je $\mu = 2$.

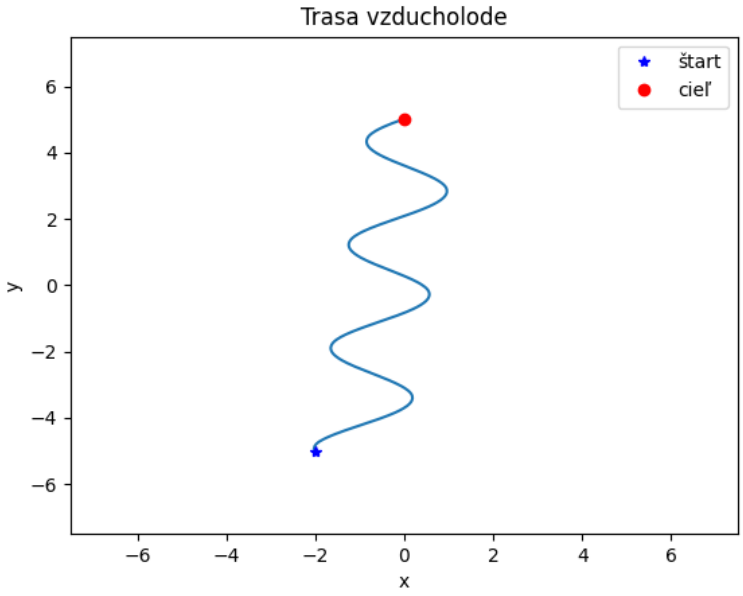
Časovo premenný vietor

Zadané veterné pole je konštantné v priestore v každom čase, ale s časom sa harmonicky mení. Demonštruje schopnosť riešiča riešiť úlohu s nestacionárnym prúdením. Zadanie je nasledovné:

- $\vec{x}_s = (-2; -5)$,
- $\vec{x}_f = (0; 5)$,
- $\vec{V}(\vec{x},t) = (2\sin(2t);\ 0)$.

Riešič vypočítal optimálnu trajektóriu úspešne a presne. Vypočítaný čas cesty je 10,08, riešič vykonal 300 krokov s adaptívnou dĺžkou kroku v rozmedzí 0,025 až 0,075. KCI je 0,076, čo je približne polovica priestorového kroku. Výpočet trval 121 sekúnd. Spočítaná trasa je vykreslená na obrázku 4.1.

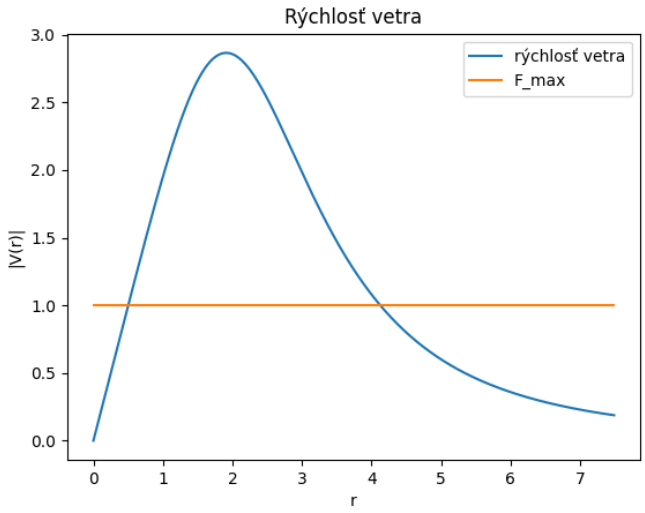
[2]Bežne presne do stredu.



Obr. 4.1 Trasa vzducholode $\vec{q}$ v časovo premenlivom vetre.

Radiálne prúdenie

Zadané je statické radiálne symetrické prúdenie s nulovou radiálnou zložkou – vzorec nižšie. Rýchlosť vetra v závislosti na vzdialenosti od stredu je zobrazená na obr. 4.2. Vzducholoď začína v strede víru – v strede symetrie prúdenia.



Obr. 4.2 Priebeh uhlovej zložky vetra v závislosti na vzdialenosti od stredu symetrie. Radiálna zložka je nulová. Horizontálna čiara je maximálna rýchlosť vzducholode.

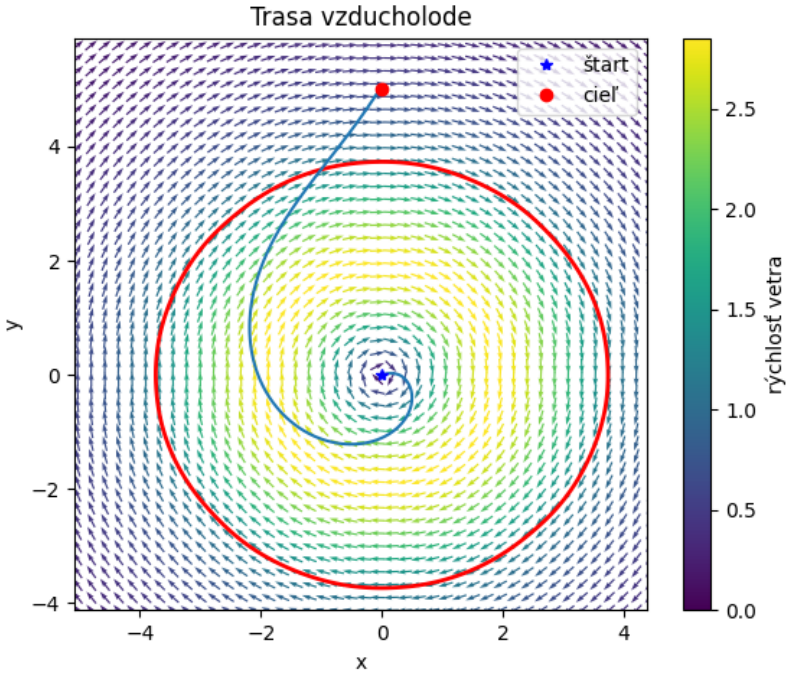
Front je v každom čase kružnica, ale trajektória vzducholode nie je priama a závisí na priebehu prúdenia. Príklad demonštruje delokalizáciu vzducholode vo fronte počas prvej fázy výpočtu. Veterné pole má vzhľadom na front len dotyčnicovú zložku a teda nevplýva na pohyb frontu, avšak ovplyvňuje pohyb vzducholode v rámci frontu. Pre rôzne priebehy prúdenia dostaneme rôzne trajektórie vzducholode s rovnakým časovým vývojom frontu a preto nie je možné

vylúčiť „neperspektívne“ časti frontu za účelom zníženia výpočtových nárokov. V tomto prípade máme dvakrát jemnejšie priestorové delenie – v každom smere 200 elementov a teda $h = 0{,}075$.

Parametre zadania sú:

- $\vec{x}_s = (0; 0)$,
- $\vec{x}_f = (0; 5)$,
- $\vec{V}(\vec{x},t) = (y, -x) \cdot \frac{1}{|\vec{x}|+0.01} \cdot \frac{2|\vec{x}|}{1+\frac{1}{40}|\vec{x}|^4}$.

Vypočítané inštrukcie sú presné – KCI je 0,035, čo je asi polovica priestorového kroku. Vypočítaný čas cesty je 4,98, výpočet trval 665 sekúnd a riešič urobil v každej fáze 510 krokov. Veľkosť časového kroku bola určená na základe maximálnej rýchlosti prúdenia, priestorového delenia a CFL pomeru ako 0,00974. Výsledok výpočtu je znázornený na obrázku 4.3.



Obr. 4.3 Vypočítaná trajektória vzducholode. Trajektória nie je priama, ale front je v každom čase kružnica. Zobrazený front v čase 3,7.

Existencia viacerých riešení

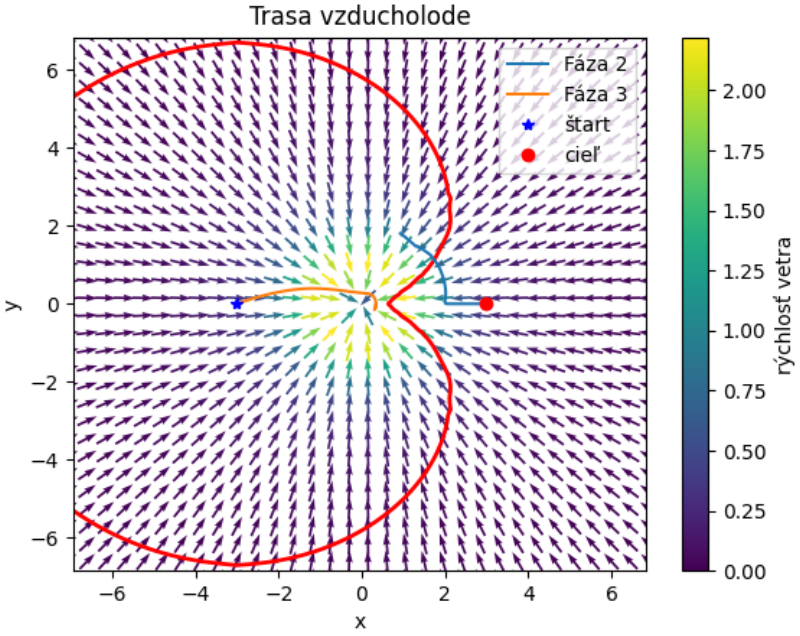
Posledný príklad na nereálny vietor demonštruje odhaliteľné zlyhanie riešiča. Zadanie má dve riešenia a front v niektorých miestach nemá definovanú normálu. Bod, v ktorom front nemá normálu, bude dosiahnutý vzducholoďou v druhej fáze. Dokonca front pretne $\vec{x}_f$ práve v tomto bode. Riešič neskončí chybovou hláškou, ale dopočíta riešenie až do konca. V mieste, kde matematicky front nemá normálu, numericky spočítaná normála smeruje do stredu medzi limitné normály z jednotlivých strán. Parametre zadania sú:

- $\vec{x}_s = (-3; 0)$,
- $\vec{x}_f = (3; 0)$,

- $\vec{V}(\vec{x},t) = -\frac{\vec{x}}{|\vec{x}|+0.01} \cdot \frac{2|\vec{x}|}{1+\frac{1}{3}|\vec{x}|^4}$.

Veterné pole je opäť radiálne symetrické, avšak oproti predošlému príkladu je jeho smer čisto radiálny a vietor smeruje do stredu. Jeho priebeh je kvalitatívne rovnaký ako v predošlom príklade, znázornený na obrázku 4.2. Podstatná je skutočnosť, že vzducholoď nemá dostatočnú rýchlosť na to, aby unikla zo stredu prúdenia[3]. Zároveň začína dostatočne ďaleko od stredu na to, aby sa dokázala vyhnúť vtiahnutiu do stredu a dosiahnuť cieľ. Vzducholoď musí nebezpečnú oblasť obísť. Keďže $\vec{x}_s$ a $\vec{x}_f$ sú stredovo súmerné podľa stredu prúdenia, existujú dve riešenia a riešič zlyhá. Spočítaná trajektória je znázornená na obrázku 4.4. V druhej fáze je numerická normála v mieste, kde matematicky normála neexistuje, presne medzi jednostrannými limitami. Preto vzducholoď sleduje nehladké miesto frontu. Chybne spočítané kormidlo sa následne prejaví v tretej fáze.

Výpočet trval 155 sekúnd, riešič urobil 406 krokov, dokým front dosiahol cieľ. Veľkosť časového kroku bola vypočítaná ako 0,023. KCS je 4,32, KCI je 2,65. Veľkosť indikátorov chyby je omnoho väčšia ako h a pohybuje sa v rádoch $|\vec{x}_f - \vec{x}_s|$, čo je jasný signál, že výsledky výpočtu nie sú použiteľné. Čas cesty bol vypočítaný ako 9,46, to stále môže byť dôveryhodné číslo, pretože je to čas, kedy front dosiahol cieľ a chyba riešiča vznikla až neskôr, v druhej fáze[4]. Nepoužiteľnosť výsledkov výpočtu vieme v tomto prípade odhaliť a je zrejmá z KCI, ktorá je súčasťou uloženého výstupu.



Obr. 4.4 Vypočítaná trajektória vzducholode z fázy 2 aj 3. V ani jednej fáze vzducholoď nedosiahla druhý koniec požadovanej trasy. Vo fáze 2 je to spôsobené neexistenciou normály v bode frontu, kde sa vzducholoď nachádzala. To spôsobilo chybne určené kormidlo, ktoré následne spôsobilo zlyhanie aj vo fáze 3. Znázornený je front v čase 7,09, nehladké miesto frontu postupuje v smere osi x a dosiahne bod $\vec{x}_f$, čo je vidieť z trajektórie vzducholode z druhej fázy, ktorá ho nejaký čas sleduje.

[3]Ale v strede je oblasť, v ktorej sa môže pomerne voľne pohybovať.

[4]Tiež úspech prvej fázy znamená, že je možné dosiahnuť cieľ.

4.3 Porovnanie s riešením sústavy ODR

Náš riešič sme porovnali s prístupom predstaveným v časti 2.5 za účelom overenia jeho presnosti a konvergencie. Riešič sme testovali na príklade stacionárneho prúdenia s charakterom jednoduchého šmyku. Numerické parametre sú rovnaké ako v časti 4.2 okrem jemnosti priestorového delenia, ktorú budeme postupne zvyšovať. Začíname v čase $t_0 = 0$ a rýchlosť vzducholode je $F_{\max} = 1$. Ďalšie údaje zadania sú:

- $\vec{x}_s = (3{,}66; -1{,}86)$,
- $\vec{x}_f = (0; 0)$,
- $\vec{V}(\vec{x},t) = (-y,0)$.

Sústavu ODR (2.8) možno pre toto špeciálne zadanie ďalej upravovať až na[5]:

$$\begin{aligned}\frac{\mathrm{d}x_1}{\mathrm{d}t} &= \cos\beta - x_2,\\ \frac{\mathrm{d}x_2}{\mathrm{d}t} &= \sin\beta,\\ \frac{\mathrm{d}\beta}{\mathrm{d}t} &= \cos^2\beta.\end{aligned}$$

Nepoznáme čas cesty ani počiatočnú podmienku pre tretiu rovnicu. Upravovaním tejto sústavy možno dostať nelineárnu sústavu dvoch algebraických rovníc, ktorých riešením je $\beta(T_f)$ a $\beta(t_0)$. Túto sústavu je potrebné riešiť numericky – použitá je Newtonova metóda. Možno tiež odvodiť rovnicu

$$\tan\beta(t) - \tan\beta(T_f) = t - T_f,$$

z ktorej získame čas cesty dosadením t_0 za t a tiež z nej získame priebeh natočenia vzducholode. Následne so znalosťou T_f vieme analyticky vyriešiť rovnice pre trajektóriu. Z uvedeného vyplýva, že vďaka takmer úplnej eliminácii numerických faktorov v tejto technike je spočítané referenčné riešenie veľmi presné.

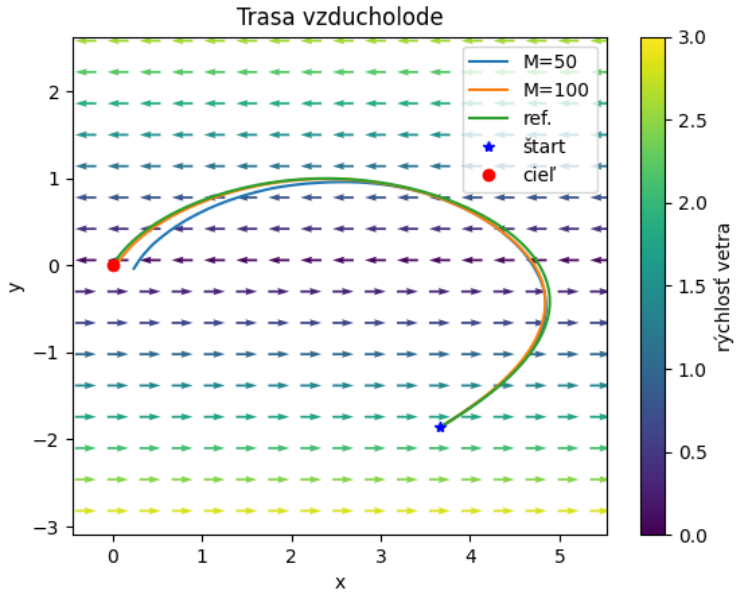
Referenčné riešenie je $\beta(0) = 1{,}83293$, $\beta(T_f) = 4{,}18847$ a $T_f = 5{,}45787$ (možno nájsť v [5]). Referenčnú trajektóriu vyjadrenú analyticky pomocou numericky získaných parametrov používame vyčíslenú v časoch s krokom 0,001. Budeme zjemňovať delenie nášho riešiča a sledovať priebežnú odchýlku numerickej trajektórie od referenčnej a celkovú odchýlku ako integrálny priemer veľkosti priebežnej odchýlky, označme celkovú odchýlku $|\Delta\vec{q}|$. Rozdiel trajektórií určíme pomocou lineárnej interpolácie nasledovne. Pre každý čas t_n interpolujeme referenčnú trajektóriu z dvoch susedných bodov do času t_n a vypočítame rozdiel interpolovaného bodu referenčnej trajektórie a $\vec{q}_n$.

Pre rôzne jemné siete $M \times M$ elementov sme riešili popísanú úlohu a sledovali konvergenciu k referenčnému riešeniu. Celkový pohľad na spočítanú trajektóriu (obr. 4.5) ukazuje, že všetky numerické výsledky sú blízko referenčnému. Zobrazujeme len trasu pre $M = 50$ a $M = 100$, pretože presnejšie výsledky by na obrázku neboli aj tak odlíšiteľné. Priebežná odchýlka so zjemňovaním delenia klesá, ako je vidieť na obr. 4.6. V nasledovnej tabuľke 4.1 sú zhrnuté výsledky pre celkovú odchýlku. Pozorujeme konvergenciu numerickej trajektórie k referenčnej.

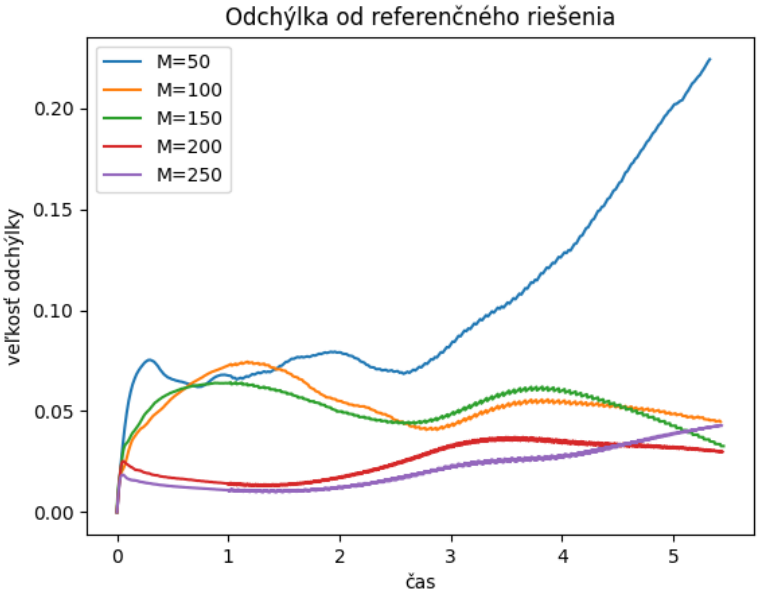
[5]Tieto výsledky sú prevzaté rovnako ako výsledky predstavené v časti 2.5.

M	$\lvert\Delta\tilde{q}\rvert$
50	0,101
100	0,053
150	0,052
200	0,025
250	0,021

Tabuľka 4.1 Celková odchýlka numerickej trajektórie v závislosti na jemnosti delenia.



Obr. 4.5 Porovnanie spočítanej trajektórie pre 50 a 100 elementov siete v každom smere s referenčným riešením.



Obr. 4.6 Priebežné odchýlky numerických trajektórií od referenčnej pre rôzne jemné delenia.

Poznámka

Uvedený príklad jednoduchého šmyku je príklad na úlohu so stacionárnym prúdením, kedy časový priebeh vzdialenosti frontu od cieľa, t.j. $\phi(\vec{x}_f,t)$, nie je monotónny. Vzdialenosť najprv rastie a od určitého času začne klesať.

4.4 Reálne dáta

V tejto časti popíšeme spôsob použitia predpovede počasia v našej práci. Úlohu riešime na obdĺžnikovej sieti, ktorej elementy sú štvorce[6]. Budeme rozlišovať tri súradné sústavy: sieťové kilometre, teda súradnice na sieti s rovnomerným delením, mapové súradnice, teda pixely na obrazovke grafického rozhrania riešiča, a sférické GPS súradnice. Predpoveď počasia poskytuje Open-Meteo.com[7] a získavame ju z internetu pomocou Python modulu `openmeteo_requests`. Open-Meteo.com v základnej verzii poskytuje obmedzený počet volaní[8], ktorý ale pre naše použitie nepredstavoval výrazné obmedzenie.

Získavanie dát o vetre

Open-Meteo.com poskytuje predpoveď počasia pre vietor vo výške nad povrchom 10m, 80m, 120m a 180m, používame predpoveď pre výšku 180m. Obdobie, na ktoré je predpoveď vydaná, nie je zásadne obmedzené a je možné získať aj minulé dáta. Modely, z ktorých Open-Meteo.com čerpá dáta, majú v našej oblasti priestorové rozlíšenie 15km a časové rozlíšenie 1 hodinu (uvedené v dokumentácii Open Meteo [6]). Jedna požiadavka obsahuje dáta po hodinách pre niekoľko[9] priestorových bodov. Ak zadáme bod, ktorý nie je uzlovým bodom siete v Open-Meteo.com, dostaneme interpolované hodnoty. Nemá preto zmysel získavať surové dáta s jemnejším priestorovým krokom ako 15km. Open-Meteo.com používa GMT[10].

Keďže počet volaní je obmedzený, naprogramovali sme sťahovač, ktorý v prípade požiadavky na viac ako 500 bodov dávkuje volania po minútach, aby nebol porušený limit na počet volaní[11]. Z rovnakého dôvodu ukladáme získané predpovede do súboru a sťahovač umožňuje voliť medzi volaním na novú predpoveď a čítaním uloženej predpovede. Každá uložená predpoveď obsahuje v hlavičke okrem parametrov volania aj časovú pečiatku volania.

Volanie do Open-Meteo.com má nasledovné parametre: súradnice bodov, v ktorých chceme predpoveď získať, časový interval v dňoch a zoznam žiadaných veličín. Implementovaný sťahovač je funkcia, ktorá vráti predpoveď ako 3D pole (dve

[6] Musia to byť štvorce kvôli FMM.

[7] `https://open-meteo.com/`

[8] Najviac 600 za minútu, 5000 za hodinu a 10000 za deň. Jedno volanie obsahuje časový vývoj žiadanej veličiny po dobu niekoľkých dní v jednom bode.

[9] 50 bodov, obmedzenie je dané tým, že príkaz do volania sa odosiela ako URL a URL má obmedzenú dĺžku. Pre účely limitu počtu volaní Open-Meteo.com považuje každý bod v priestore za samostatné volanie.

[10] Preto absolútny čas v tomto riešiči je v práci GMT.

[11] Pri porušení limitu dostaneme chybu z modulu `openmeteo_requests`, čomu sa chceme vyhnúť.

súradnice v priestore a jedna v čase) spolu s časovou pečiatkou predpovede[12]. Na vstupe má parametre siete, v ktorej uzloch získava predpoveď, časový interval v dňoch a inštrukcie pre použitie a ukladanie predpovedí v súboroch.

Projekcia zo sféry na rovinu

Používame obdĺžnikovú sieť s rovnomerným priestorovým delením v sieťových kilometroch. Rozmery siete sú 845km × 585km. Priestorový krok[13] volíme rádovo 10km. Konštantný priestorový krok na sieti považujeme za konštantný aj v GPS súradniciach. To znamená, že používame afinný prepočet sieťových kilometrov na GPS súradnice a teda sieť má konštantný priestorový krok aj v zemepisných súradniciach (φ,λ), ktorý ale nie je konštantným priestorovým krokom na sfére. Zakrivená plocha skutočného sveta je pre model rovina s rovnomerným delením. Použitá projekcia je volená tak, aby bola presná približne v strede použitej mapy, teda na súradniciach 50,479N, 16,559E, tento bod je zároveň počiatok v sieťových kilometroch. Pri tejto voľbe zodpovedá 1° zemepisnej šírky 111,317km a 1° dĺžky je 70,837km. Na severnom okraji mapy, kde je chyba projekcie najväčšia, zodpovedá 1° dĺžky[14] 67,837km, čo je relatívna chyba 4,6%. Chyba je pre naše účely prijateľná.

Spracovanie dát o vetre

Predpoveď počasia získavame v uzloch siete, ktorá má v sieťových kilometroch zhodné rohy s výpočtovou sieťou a jej delenie je rovnomerné s krokom 15km, máme teda necelých 3000 uzlov[15]. Sieťové kilometre sa pred volaním transformujú na GPS súradnice. Získanú predpoveď interpolujeme bikubicky na výpočtovú sieť. Predpoveď dostaneme v každej celej hodine daného dňa. V čase používame po častiach afinnú interpoláciu, ktorou na základe vetra v susedných celých hodinách T_k,T_{k+1} získame vietor pre ľubovoľný čas[16] $t \in [T_k,T_{k+1})$ vzťahom $\vec{V}(\vec{x},t) = (1-\Delta t)\cdot\vec{V}(\vec{x},T_k) + \Delta t\cdot\vec{V}(\vec{x},T_{k+1})$, kde $\Delta t = (t-T_k)/\mathrm{h}$. Interpoláciu vykonávame pri každej aktualizácii veterného poľa počas výpočtu a adaptívny časový krok sa určuje vždy podľa aktuálneho interpolovaného vetra.

Mapa

Používame mapu OpenStreetMap[17]. Použitá časť obsahuje Česko a časti blízkych štátov. Prevod GPS súradníc na mapové sa uskutočňuje pseudo-rovníkovou

[12]Skutočný čas získania predpovede – odlíšime takto aktuálnu predpoveď od predpovede načítanej zo súboru.

[13]Aby boli elementy siete štvorce a zároveň predpoveď počasia získavaná po 15km (to nemôžeme meniť, aby sme mohli používať uložené dáta) obsahovala rohové uzly, musí byť každý rozmer siete celočíselným násobkom 15km aj h. To je splnené napríklad pre $h = 15\mathrm{km}$, $h = 9\mathrm{km}$.

[14]V zemepisnej šírke sú sieťové kilometre presné v celej sieti.

[15]Čakanie na stiahnutie predpovede je teda 5 minút.

[16]$T_{k+1} - T_k = 1\mathrm{h}$, tu h neznačí priestorový krok (h), ale hodinu (h).

[17]`https://www.openstreetmap.org/#map=7/49.817/15.478`

projekciou, ktorá je štandardná pre digitálne mapy. Transformačné vzťahy sú:

$$X = \frac{1}{2\pi} \cdot 2^{\text{zoom}} \cdot (\pi + \lambda),$$
$$Y = \frac{1}{2\pi} \cdot 2^{\text{zoom}} \cdot (\pi - \ln \tan(\frac{\pi}{4} + \frac{\varphi}{2})).$$

Súradnice (X,Y) následne transformujeme afinne tak, aby mapové súradnice mali počiatok v SZ rohu mapy a os y smerovala nadol[18]. Prevod medzi mapou a GPS je ladený podľa trojmedzí CZ-PL-DE a CZ-PL-SK. Keďže mapové súradnice nie sú afinnou funkciou GPS súradníc, konštantný priestorový krok sieťových km nie je konštantný na obrazovke programu[19].

Používateľské rozhranie

Vytvorili sme používateľské rozhranie umožňujúce intuitívne zadať parametre úlohy a vykonať výpočet. Tak ako všetky súčasti práce, aj toto je implementované v Pythone. Umožňuje zadať rýchlosť vzducholode, čas štartu, maximálny prípustný čas letu, východzí a cieľový bod. Ďalej indikuje priebeh výpočtu a zobrazuje jeho hlavné výstupy. Ďalej je možné zadať názov súboru, do ktorého sa ukladajú výsledky a pokyny pre sťahovač predpovede počasia – umožňuje rozhodnúť, či sťahovať z internetu novú predpoveď, použiť predpoveď zo súboru alebo prípadne pokúsiť sa stiahnuť predpoveď a v prípade neúspechu použiť predpoveď zo súboru. Tiež je možné uložiť stiahnutú predpoveď alebo stiahnuť predpoveď bez vykonania výpočtu. Podrobnosti sú uvedené v nápovede k programu, ktorá je súčasťou používateľského rozhrania, a tiež v súbore `README.txt`. Program, ako aj riešič pre umelý vietor a ukážkové výstupy, sa nachádza v prílohe práce a tiež je dostupný na stránke autora práce[20]. Po skončení výpočtu sa vykreslí spočítaná trasa a je možné zobraziť dôležité výstupy (navigačné inštrukcie a údaje o výpočte, ktoré sa ukladajú) a tiež výpočtovú sieť, adaptívny časový krok, priebeh vzdialenosti frontu od cieľa, vietor a ďalšie. Po skončení výpočtu sa môže prehrať animácia letu vzducholode vrátane zobrazenia vetra. Ukážka grafického rozhrania je na obr. 4.7.

Zhrnutie a záverečné poznámky

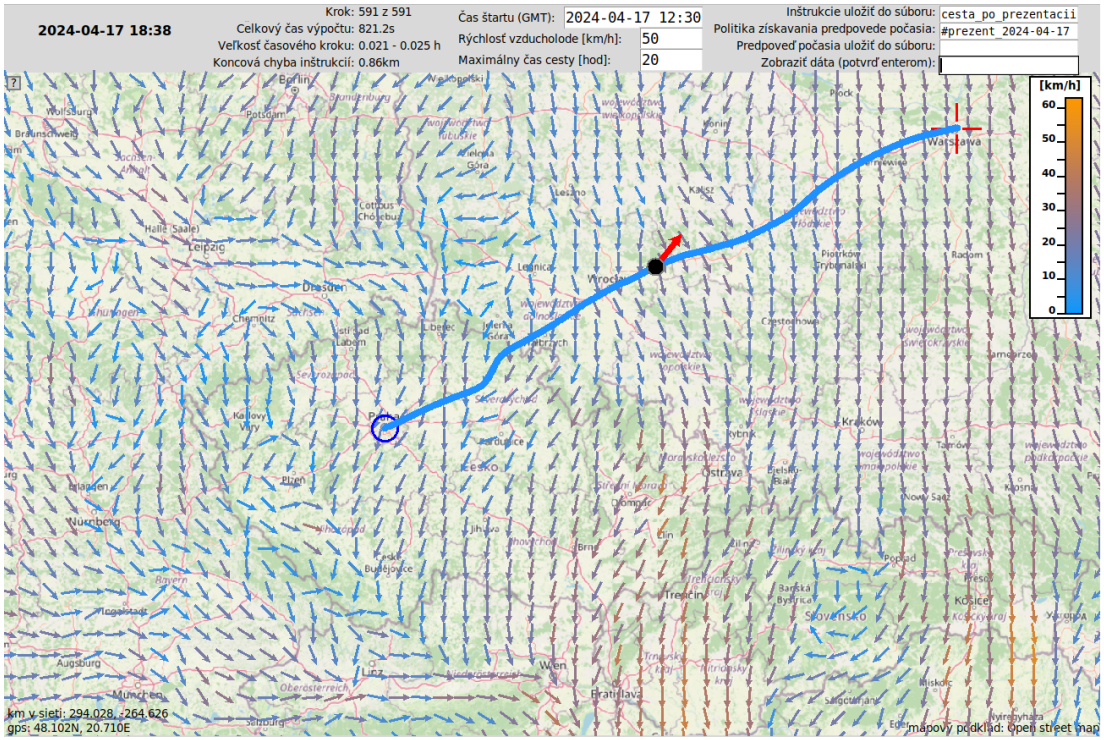
Program na riešenie Zermelovej úlohy pre reálne dáta je funkčný, vyladený a spoľahlivý. Jeho jediný výraznejší nedostatok je, že vyžaduje inštaláciu riešiča Firedrake. Presnosť výpočtov je dobrá.

V kóde je vyznačené, kde sa nastavuje CFL parameter, priestorový krok a iné významné údaje, ktoré nie je možné nastaviť cez grafické rozhranie. Tiež všetky premenné, s ktorými program pracuje, sú prístupné cez príkazový riadok. Program ďalej obsahuje vykresľovacie funkcie pre jednoduché generovanie grafov z výstupov výpočtu. To isté platí pre riešič pre umelý vietor.

[18]To je prirodzené pre vykresľovanie v moduli `tkinter`, v ktorom sme vytvorili grafické rozhranie.

[19]V smere y. V smere x je delenie rovnomerné.

[20]`https://simkovip.github.io/V%C3%BDpo%C4%8Det%20navig%C3%A1cie%20vzducholode.zip`



Obr. 4.7 Používateľské rozhranie riešiča úlohy pre reálne dáta.

5 Známe nedostatky a diskusia

Vytvorený riešič dokáže riešiť umelé aj realistické zadania s dostatočnou presnosťou. V prípade, že výpočet je nepresný, vieme nepresnosť zistiť z výstupov programu. V čase písania tejto práce si je autor vedomý niektorých nedostatkov a možných zlepšení riešiča, ktoré tu zhrnieme a navrhneme možný ďalší vývoj. Riešenie spomenutých nedostatkov je nad rámec tejto práce. Tiež uvedieme diskusiu predpokladov úlohy, ich možných oslabení a pozrieme sa na riešenú problematiku v širšom kontexte.

Určenie frontu vo FMM

Najväčším známym nedostatkom je spracovanie levelsetu vo FMM blízko frontu. Front je identifikovaný ako množina vrcholov, ktorých hodnota je v abs. hodnote menšia ako tolerancia a tieto vrcholy sa použijú ako východzie, s tým, že ich hodnota sa nezmení. Vo výsledku FMM necháva deformovaný levelset v okolí frontu, čiže v mieste, kde je presnosť najviac potrebná. Z hľadiska presnosti výpočtu sa to neukázalo komplikujúce – aj napriek tejto vade výpočet dosahuje úctyhodnú presnosť. FMM plní svoj účel v tom, že riešič prvej fázy nezlyháva a konverguje po troch iteráciách. Avšak deformácia levelsetu v okolí nulovej úrovňovej množiny znemožňuje použitie presnejších metód, ktoré sa práve na levelset v okolí frontu spoliehajú, napríklad priebežnej projekcie v druhej fáze. Tiež zhoršuje výpovednosť dát a indikátorov získavaných v priebehu výpočtu z levelsetu v blízkosti frontu, napríklad priebežnej chyby v druhej fáze. V dostatočnej vzdialenosti od frontu, kde FMM spracuje levelset správne, tieto problémy nenastávajú, nakoľko oblasť deformácie okolo frontu je malá.

Ako možnosť riešenia navrhujeme pri inicializácii FMM na základe hodnôt vrcholov blízkych frontu interpolovať hodnoty levelsetu na celom okolí frontu. Interpolovaným levelsetom odhadnúť polohu frontu a na základe nej potom preškálovať hodnoty uzlov blízkych frontu tak, aby predstavovali znamienkovú vzdialenosť. Následne by prebehla FMM tak, ako je implementovaná.

Rád metód druhej a tretej fázy

Ďalším nedostatkom je nízky rád metód použitých v druhej a tretej fáze. Problém súvisí s predošlým. Z prvej fázy máme uložený levelset v každom čase a adaptívny časový krok, ale nie sme na toto delenie odkázaní, pretože môžeme interpolovať hodnoty do ľubovoľného času a tým získať levelset pre výpočet kormidla v jemnejšom časovom delení v druhej fáze (ktorej výpočet trvá kratšie a preto si menší časový krok môžeme dovoliť). To isté môžeme urobiť s kormidlom z druhej fázy a výpočtovo rýchlejšiu tretiu fázu vykonať s jemnejším časovým delením. Dosiahneme tak väčšiu presnosť výstupov a zlepšíme výpovednosť KCI, napríklad metódou Runge-Kutta 4 alebo niektorou implicitnou metódou. Avšak presnosť interpolácie levelsetu závisí opäť na tom, ako presne levelset zodpovedá znamienkovej vzdialenosti v blízkosti frontu. Preto je potrebné najprv vyriešiť problém identifikácie frontu vo FMM.

Optimalizácia výpočtovej náročnosti

Informácie poskytované levelsetom ďaleko od frontu nemajú pre úlohu veľkú hodnotu. Levelset definovaný na celej oblasti zvyšuje dimenziu problému a tým výpočtové nároky. Riešením spomenutým v práci [2] je počítať vývoj levelsetu na pohyblivom úzkom páse okolo frontu. To by výrazne znížilo trvanie výpočtu prvej fázy. V prípade našej implementácie vo Firedrake by si to vyžiadalo veľké zmeny v kóde prvej fázy. V súvislosti s tým prichádza do úvahy voľba adaptívneho časového kroku, ktorý sa zbytočne počíta z maximálnej rýchlosti vetra na celej oblasti. Ak by sme riešili vývoj levelsetu na menšej množine, v niektorých prípadoch by sa tým vylúčili lokálne rýchle časti prúdenia vzdialené od frontu. Časový krok by mohol byť väčší pri zachovaní presnosti. Tak isto v druhej a tretej fáze by bolo možné určovať veľkosť časového kroku len na základe vetra v okolí aktuálnej polohy, čím by sa znovu urýchlil výpočet. Predtým ale musíme vedieť spoľahlivo interpolovať dáta z predošlých fáz (pretože by sme mali rozdielny časový krok), čo stojí na presnosti FMM v blízkosti frontu.

Ďalšie možné vylepšenia

Súčasná implementácia využíva riešič Firedrake. Pre účely komerčného využitia je potrebné implementovať niektoré použité metódy[1] bežnejšími prostriedkami, napr. pomocou knižnice `scipy`. To by si vyžiadalo veľké úsilie, ktorého prínos je v čase písania tejto práce otázny.

Predpoklady na prúdenie

V práci predpokladáme, že vertikálny profil vetra je konštantný – riešime dvojrozmerný problém. Dôvody na to sú čisto praktické. Zvýšenie dimenzie o 1 by zvýšilo náročnosť výpočtov. Získavanie realistických dát o vetre pre rôzne výšky nad povrchom je viac problematické. Aj keď v špecifických prípadoch by prinieslo vertikálne rozlíšenie nové zistenia (napríklad pri lietaní cez horské oblasti), dvojrozmerná aproximácia je pre túto prácu dostatočná. Úlohu sme síce riešili pre navigáciu vzducholode, výsledky sú však použiteľné aj pre plavbu po mori.

Ďalším zjednodušením je, že predpoveď počasia považujeme za presnú. Sledovanie citlivosti riešenia na zmeny prúdenia by prinieslo ďalší indikátor spoľahlivosti výstupov. Ďalšou možnosťou, ako zlepšiť program, je priebežne aktualizovať výpočet na základe novej predpovede počas letu.

Kritériá optimality

Trajektóriu sme optimalizovali pre čas cesty. Implementovali sme efektívny algoritmus na výpočet takých inštrukcií, ktoré dostanú vzducholoď do cieľa v najkratšom možnom čase. To je priamočiara úloha s jasným praktickým uplatnením. Existujú však iné varianty pre let vzducholode. Neuvažovali sme napríklad možnosť zakotvenia počas letu. Vzducholoď by prečkala nepriaznivé prúdenie pripútaná k povrchu. Špeciálne, úloha by sa dala riešiť s tým, že sa vypočíta nielen optimálna sada inštrukcií, ale aj optimálny čas štartu v zadanom intervale – táto varianta má význam aj pre plavbu po mori.

[1] Metóda konečných prvkov, viacrozmerná Newtonova metóda, riešič lineárnych sústav.

Za optimálne riadenie považujeme také, ktoré minimalizuje čas cesty. Na to sa snažíme využívať prúdenie tak, aby nám pomohlo dostať sa do cieľa (alebo skôr zisťujeme, ako veľmi sa dá prúdenie využiť). Pritom sme ukázali, že optimálne je, aby sa vzducholoď celý čas pohybovala plnou rýchlosťou. Ponúka sa otázka, či je možné dostať sa do cieľa hlavne s pomocou prúdenia a s minimálnym zásahom vlastného pohonu vzducholode. Teda môžeme optimalizovať nie čas cesty, ale spotrebu paliva. Výstupom riešiča by popri inštrukciách, ktoré by zahŕňali okrem smeru aj veľkosť rýchlosti vlastného pohonu, bolo aj množstvo paliva potrebné na cestu (prípadne, ako sme videli, nemožnosť dosiahnuť cieľ). Spolu s maximálnym časom by užívateľ zadal aj rozpočet, teda maximálnu spotrebu paliva. Zrejme sme tým pridali úlohe stupeň voľnosti. Riešič zohľadňujúci spotrebu paliva by mal väčší prínos ako riešič optimalizujúci čas cesty.

Širší kontext

Implementovaný riešič môže nájsť uplatnenie aj mimo zamýšľané oblasti. Propagácia frontu dosiahnuteľnosti riešená v prvej fáze rieši úlohu, kam až sa môže v danom prúdení dostať objekt s obmedzenou vlastnou rýchlosťou. V prípade, že naším zámerom nie je letieť z miesta štartu do cieľa, ale vypátrať stratené plavidlo, prípadne odhadnúť oblasť, kde sa môže vyskytovať, je vyvinutý riešič pripravený slúžiť aj tomuto účelu. Pritom front dosiahnuteľnosti nemusí reprezentovať všetky možné smerovania hľadaného objektu. Ak poznáme približne smer vlastného pohonu objektu, vývojom frontu môžeme reprezentovať neistotu a prípadné odchýlky tohto smeru a získať presnejšie a výpovednejšie dáta. Otvára sa možnosť sledovať citlivosť trajektórie na malé zmeny riadenia.

Záver

Vytvorili sme funkčný riešič Zermelovej úlohy. Dokážeme spočítať navigačné inštrukcie pre rôzne prúdenia. Riešič dokáže spracovať aj dáta z reálnej aktuálnej predpovede počasia a počítať inštrukcie pre skutočné zadania. Vyvinuli sme intuitívne používateľské prostredie pre prácu s riešičom pri riešení reálnych úloh, s automatickým sťahovaním dát, ukladaním a zobrazovaním výstupov. Prostredie sa charakterom blíži plnohodnotnému programu. Výpočet je aj napriek použitiu priamočiarych techník rýchly a presný. Typicky trvá riešenie úlohy do desať minút. Koncová chyba inštrukcií, hlavný použitý indikátor presnosti vyjadrujúci presnosť, s akou dosiahne vzducholoď cieľ, dosahuje hodnôt až desatiny priestorového kroku. Pri riešení reálnych úloh to predstavuje minutie cieľa o nanajvýš niekoľko kilometrov pri letoch dlhých stovky kilometrov.

Druhá a tretia fáza výpočtu si žiada vylepšenia. Najmä identifikácia a oprava hodnôt levelset funkcie v blízkosti frontu dosiahnuteľnosti pri reinicializácii v prvej fáze výpočtu je slabá a zanecháva vo funkcii deformácie. Následky nedostatku spočívajú, viac ako v nepresnostiach výpočtu, skôr v nemožnosti použiť presnejšie metódy v neskorších fázach výpočtu. Napriek tomu dosahujú výsledky úctyhodnú presnosť, ktorú dokážeme určiť koncovou chybou inštrukcií. V niektorých prípadoch je presnosť výpočtu malá. Ak je výpočet nepresný, je to možné zistiť práve z koncovej chyby inštrukcií. Máme nástroj, ktorým vieme prípadnú nízku presnosť alebo nepoužiteľnosť výstupov odhaliť.

Ako sme uviedli v diskusii, implementovaný riešič je možné použiť aj na odlišné účely, napríklad odhadovanie polohy strateného plavidla alebo určenie citlivosti trajektórie na perturbácie riadenia.

Literatúra

1. Zermelo, E. Über das Navigationsproblem bei ruhender oder veränderlicher Windverteilung. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik* [online]. 1931, roč. 11, č. 2, s. 114–124 [cit. 2024-04-02]. Dostupné z DOI: https://doi.org/10.1002/zamm.19310110205.
2. Lolla, S. V. T. *Path planning in time dependent flow fields using level set methods* [online]. 2012. [cit. 2024-04-07]. Dostupné z : http://hdl.handle.net/1721.1/78212. Diploma thesis. Massachusetts Institute of Technology.
3. Chachuat, Benoit. *Nonlinear and Dynamic Optimization: From Theory to Practice* [online]. 2007. [cit. 2024-04-07]. Dostupné z : https://www.researchgate.net/publication/37452197_Nonlinear_and_Dynamic_Optimization_From_Theory_to_Practice. EPFL.
4. Sethian, J. A. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. New York: Cambridge University Press, 1999. Second edition. ISBN 978-0-521-64557-7.
5. Bryson, Arthur Earl; Ho, Yu-Chi. *Applied optimal control: optimization, estimation and control*. Washington: Hemisphere, 1975.
6. *Weather Forecast API, Documentation* [online]. [cit. 2024-04-24]. Dostupné z : https://open-meteo.com/en/docs/.