

Káosz az *El Niño* egy
modelljében
– számítógép által segített
bizonyítás

Indig Balázs

MÉRNÖK INFORMATIKUS MSC HALLGATÓ

TDK DOLGOZAT

Témavezető : Garay Barnabás egy.tan.

PPKE ITK

Budapest 2011 November

Tartalomjegyzék

0. ÖSSZEFOGLALÁS, BEVEZETÉS, KÖSZÖNETNYILVÁNÍTÁS	4
1. A MÖGÖTTES MATEMATIKA	6
1.1. A topologikus Smale patkó mint a determinisztikus káosz prototípusa	6
1.2. Elegendő feltétel topologikus Smale patkó létezésére	7
1.3. Az <i>El Niño</i> Vallis modellje	11
1.4. A Lorenz és a Vallis rendszer összehasonlítása	12
2. A SZÁMÍTÓGÉPES FELADAT	14
2.1. Számítógéppel segített káosz–bizonyítás	14
2.2. A célzott találgatások heurisztikája	16
2.3. Egy a posteriori becslés	22
2.4. Intervallum–aritmetika dióhéjban	26
2.5. A felhasznált eszközök	28
2.6. A program felépítése vázlatosan	30
2.7. A program részei	31
2.7.1. A <i>Branch and Bound</i> módszer	31
2.7.2. Az intervallumok felezése: a <i>Branch</i> függvény	34
2.7.3. A <i>Poincaré</i> visszatérési függvény második iteráltjának képzése	35
2.7.4. A logaritmikus keresés eljárás részletei	36
2.7.5. A logikai függvények részletei	37
HIVATKOZÁSOK/REFERENCES	41

3. A program forráskódja	44
4. Számítógépes ábrák	66

0. ÖSSZEFOGLALÁS, BEVEZETÉS, KÖSZÖNETNYILVÁNÍTÁS

Ahogy E.N.Lorenz (1963) modellje a légköri cella-áramlások egy, szinte a végtelenségig leegyszerűsített leírását adja, ugyanúgy viszonyul K.G.Vallis (1988) modellje az *El Niño* jelenségéhez.

Mind a Lorenz, mind a Vallis modell háromdimenziós, lineáris és kvadrátikus tagokat tartalmazó autonóm közönséges differenciálegyenlet. Jól ismert, hogy a paraméterek klasszikus értékei mellett mindkettő kaotikus viselkedést mutat. A Lorenz modell kaotikusságát — egészen pontosan azt, hogy a Lorenz modell háromdimenziós fázisportréja tartalmazza a káosz tankönyvi prototípusának számító Smale patkó egy példányát (és így legalább annyira kaotikus mint a Smale patkó) — K.Mischaikow és M.Mrozek (1995) bizonyították elsőként, intervallum-aritmetika és számítógép segítségével.

A jelen TDK dolgozat az általuk kidolgozott eljárás megisméltése a Vallis modellre (az azóta eltelt 15 évben elérhetővé vált egyszerűsítésekkel), más szóval a Vallis modell kaotikus voltának matematikai értelemben vett bizonyítása. Mindehhez a keretet az N.Nedialkov által létrehozott intervallum-aritmetikai VNODE-LP (Validated Numerical Ordinary Differential Equations - Literate Programming) C++ programkönyvtár biztosítja. A beágyazott Smale patkó megkereséséhez szükséges a priori információ — amely alapvető feltétele a VNODE-LP programkönyvtár hatékony használatának — célzott találgatások, a MATLAB grafikus lehetőségeit kihasználó interaktív, számítógépes heurisztika eredménye.

A TDK dolgozat lehetősége a témavezetőm által 2011 tavaszán tartott "Számítógéppel segített káosz-bizonyítások" tantárgy foglalkozásain születt meg. A hetente visszatérő alkalmak során Fiáth Péter hallgató-társam és jómagam a Vallis modellnek a Lorenz rendszerére emlékeztető fázisportréja alapján kiválasztottuk a $z = 0.3$ egyenletű síkot, amelyen a (későbbiekben tárgyalásra kerülő) (K) Következménynek megfelelő halmaz-elrendezést számítógépes kísérletezéssel, különféle heurisztikus módszerek, alapvetően egydimenziós közelítések segítségével próbáltuk meghatározni. A munkának ez a része sokban hasonlított egy háromdimenziós alakfelismerési feladathoz. A geometriai problémák számítógépes kezelésében az azóta mérnök informatikus MSc diplomát szerzett Fiáth Péter korábbi alakfelismeréses tapasztalatai meghatározó szerepet játszottak. Mivel jómagam az ELTE programtervező informatikus alapszakán kizárólag algebrai és kombinatorikus jellegű programozási feladatokkal foglalkoztam, a geometriai szemléltetés és intuíció

területén sokat kellett tanulnom Pétertől, aki mindig önzetlenül volt segítségemre.

A tényleges bizonyítást, immár Fiáth Péter közreműködése nélkül, az ősz folyamán végeztem el. Ez alapvetően programozási feladat volt, amelynek lényegét a 2.1 alpontban vázolom. A menet közben felmerülő tapasztalatok szükségessé tették a két kritikus halmaz egyikének, a MATLAB heurisztikával nyert R halmaz újra-definiálását. A sikerre vezető heurisztikát a MATLAB és a sokkal hatékonyabb GSL (GNU Scientific Library 1.15) C++ programrészletek együttes alkalmazása tette lehetővé.

A témavezetőm és szegedi munkatársai által írt dolgozat [4] számítástechnikai része szintén a VNODE programcsomagot használja. Az ottani

$$\ddot{x} + \frac{1}{10} \dot{x} + \sin(x) = \cos(t)$$

inga-egyenletben a Poincaré metszősík a $t = 0$ idősík, a visszatérési idő pedig a periodicitásnak megfelelően a 2π , azaz egy állandó. Az *El Niño* Vallis modelljében a Poincaré metszősík meghatározása része a heurisztikának, a visszatérési idő függ a kezdeti értéktől, valamint a korábbinál finomabb a posteriori becslésre volt szükség.

Köszönetnyilvánítás: *Fiáth Péter hallgató-társam és a témavezetőm mellett köszönetet szeretnék mondani Koller Miklós, Tuza Zoltán valamint Zsedrovits Tamás doktoranduszoknak, akik biztosították a Práter utcai épület harmadik emeletén lévő saa-r számítógéphez, a Kar legnagyobb számítógépéhez való hozzáférést a számomra.*

Ezen a nyolcmagos, 3.4GHz-es gépen a tényleges számítógépes bizonyítás kicsivel több, mint két napig tartott.

1. A MÖGÖTTES MATEMATIKA

A számítógéppel segített káosz-bizonyítások megjelenése kiemelkedő eseménye volt a dinamikus rendszerek történetének. A legfontosabb eredmények között van két olyan is, amelyik a

$$\left. \begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= rx - y - xz \\ \dot{z} &= xy - bz \end{aligned} \right\} \quad (1)$$

Lorenz rendszerrel — a klasszikus paraméterértékek $\sigma = 10$, $b = \frac{8}{3}$, $r = 28$ — kapcsolatos: ezek Mischaikow és Mrozek (1995) [12] bizonyítása¹ a topologikus Smale patkó, valamint Tucker (1999) [18] bizonyítása az ú.n. különös attraktor létezéséről.

1.1. A topologikus Smale patkó mint a determinisztikus káosz prototípusa

A következő bekezdés tartalmazza a topologikus Smale patkó definícióját².

Legyenek L és R egy (X, d) metrikus tér nem üres, kompakt és diszjunkt részhalmazai (például a sík egymást nem metsző két négyszög-lemeze). Legyen továbbá $\varphi : L \cup R \rightarrow X$ folytonos leképezés. Azt mondjuk, hogy $\{\gamma_k\}_{k \in \mathbb{Z}}$ mindkét irányban végtelen $0 - 1$ sorozat, ha $\gamma_k \in \{0, 1\}$ minden $k \in \mathbb{Z}$ esetén. Ha $Q_0 = L$, $Q_1 = R$, az index-sorozat pedig mindkét irányban végtelen $0 - 1$ sorozat, akkor $\{Q_{\gamma_k}\}_{k \in \mathbb{Z}}$ mindkét irányban végtelen $L - R$ halmazsorozat. A φ leképezés az $L \cup R$ halmazon $L - R$ értelemben kaotikus, ha a tetszőlegesen választott mindkét irányban végtelen $\{Q_{\gamma_k}\}_{k \in \mathbb{Z}}$ $L - R$

¹Mischaikow és Mrozek 1995-ös cikkében [12], [13] a $\sigma = 45$, $b = 10$, $r = 54$ paraméterek szerepeltek. A bizonyítás számítógépes részének kiterjesztése a klasszikus paraméterekre további hosszú hónapokig tartott, és egy új szerzőtárs közreműködését igényelte [14]. Az (1) Lorenz rendszer $\text{div}(1) = -\sigma - 1 - b$ divergenciája a helytől független állandó, amely a kaotikus paramétertartományban csak negatív lehet. Növekvő divergencia, más szóval a(z attraktor nulla-mértékűségét biztosító) térfogat-összehúzódás csökkentése mellett a numerikus vizsgálatok határozottan nehezebbé válnak. A -56 [12] és a $-\frac{41}{3}$ [14] után a divergencia (egyre gyengébb térfogat-összehúzódást jelentő) $\text{div}(1) = -5$ választása ebben a TDK dolgozatban — figyelembe véve a számítógépek sebességének az elmúlt 15 év során megtapasztalt állandó növekedését — már nem igényelt átlag feletti kapacitásokat.

²a szűkebb értelemben vett matematikáról szóló mondatokban szó szerint követem a témavezetőmtől készen kapott megfogalmazásokat.

halmazsorozathoz létezik olyan mindkét irányban végtelen $\{x_k\}_{k \in \mathbb{Z}} \subset L \cup R$ pontsorozat, hogy $x_{k+1} = \varphi(x_k)$ és $x_k \in Q_{\gamma_k}$ minden $k \in \mathbb{Z}$ esetén.

Jól ismert, hogy Smale klasszikus példája, a patkó-leképezés az ottani L és R téglalapok unióján rendelkezik a fenti tulajdonsággal. Ezért az $L - R$ káoszt szokás Smale–patkó típusú káosznak is nevezni, vagy más kifejezéssel élve topologikus Smale patkónak. Ebben a dolgozatban káosz alatt mindig $L - R$ káoszt értek. Amint azt a 4. Ábra is mutatja, a topologikus Smale patkó elnevezés a szemléletesebb, de az $L - R$ káosz elnevezés (left–right) jobban utal mind a tényleges matematikai tartalomra, mind a számítógéppel segített káosz–bizonyítások szokásos menetére.

Az $L - R$ káosz definíciója alapján le lehet vezetni egy olyan $\Lambda \subset L \cup R$ halmaz létezését, amelyet a φ leképezés önmagába visz, s amelyen a periodikus pályák halmaza sűrű, a periodusok hosszúsága nem korlátos, amelyen létezik sűrű pálya, és φ folytonossága érzékeny. Itt az "érzékeny" szó szakki-fejezés, amely a kaotikus halmazok legfeltűnőbb tulajdonságára utal: létezik olyan $\eta_0 > 0$, hogy minden $\delta > 0$ és minden $x \in \Lambda$ esetén van olyan $\tilde{x} \in \Lambda$ és $k \in \mathbb{N}$, hogy $d(x, \tilde{x}) \leq \delta$ de a k -adik iteráltakra $d(\varphi^k(x), \varphi^k(\tilde{x})) \geq \eta_0$. Ezt a tulajdonságot szokás a "ha egy lepke meglibbenti a szárnyát Alaszkában, akkor abból tornádó lesz Texasban" hasonlattal (butterfly effect) érzékeltetni. A hasonlat abból a szempontból tökéletes, hogy világosan utal a hosszabb távú időjárás-előrejelzések lehetetlenségére: roppant kicsiny effektusok is járhatnak hatalmas következményekkel — sőt járnak is. (Ha ez így van, akkor mi teszi lehetségessé a számítógéppel segített káosz–bizonyításokat — kérdezhetné valaki — miért nem okoznak bajt ott a kerekítések és egyéb, a számítások során elkerülhetetlenül fellépő pontatlanságok. A válasz az, hogy az $L - R$ káosz, jóllehet tele van mindenféle labilis viselkedéssel, mint szuperstruktúra nagyon is stabil.)

1.2. Elegendő feltétel topologikus Smale patkó létezésére

A legegyszerűbb ilyen feltételt maga Smale adta meg[16].

(T) **Tétel:** Legyen $X = \mathbb{R}^2$. Az $x = (x_1, x_2) \in \mathbf{R} \times \mathbf{R}$ jelöléssel, az 1. Ábrának megfelelően definiáljuk az alábbi halmazokat:

$$E = \{x \in X \mid 1 \leq |x_1| \leq 2, |x_2| \geq 2\} \ , \ O_C = \{x \in X \mid |x_1| < 1\} \ ,$$

$$R = \{x \in X \mid 1 \leq x_1 \leq 2, |x_2| \leq 2\} \ , \ O_R = \{x \in X \mid x_1 > 2\} \ ,$$

$$L = \{x \in X \mid -2 \leq x_1 \leq -1, |x_2| \leq 2\}, \quad O_L = \{x \in X \mid x_1 < -2\}.$$

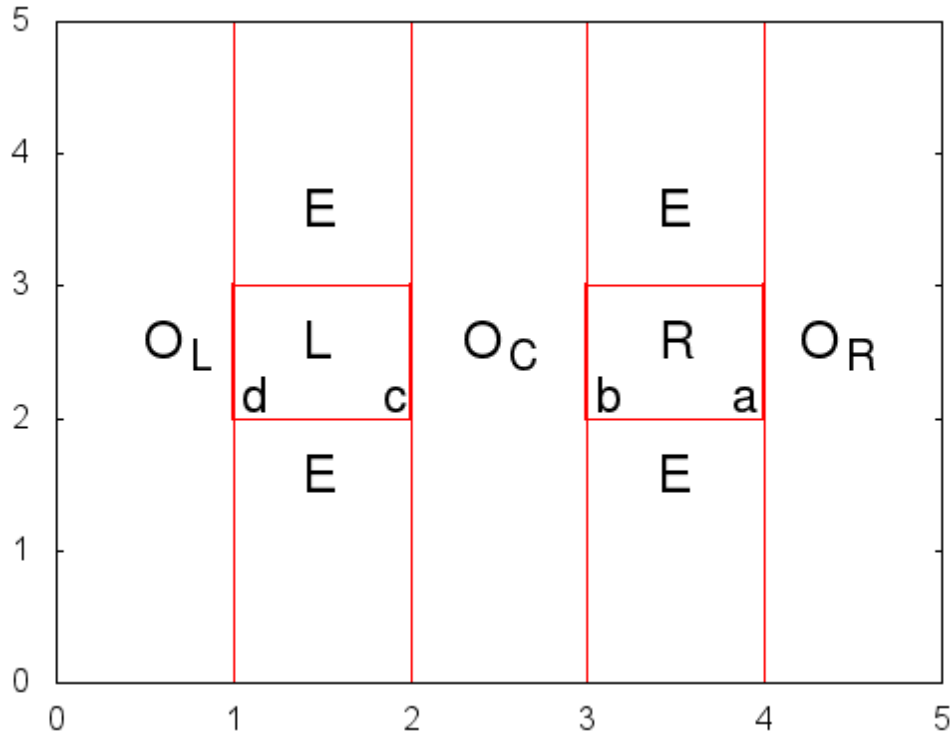
Legyenek továbbá

$$a = R \cap cl(O_R), \quad b = R \cap cl(O_C), \quad c = L \cap cl(O_C), \quad d = L \cap cl(O_L),$$

ahol $cl(O_R)$, $cl(O_C)$ és $cl(O_L)$ rendre az O_R , O_C és O_L halmazok lezárása. Végezetül legyen $\varphi : L \cup R \rightarrow X$ folytonos leképezés és tegyük fel, hogy

$$\varphi(a) \cup \varphi(c) \subset O_R, \quad \varphi(b) \cup \varphi(d) \subset O_L \quad \text{és} \quad \varphi(L) \cup \varphi(R) \subset X \setminus E. \quad (2)$$

Ekkor a φ leképezés az $L \cup R$ halmazon $L - R$ értelemben kaotikus.

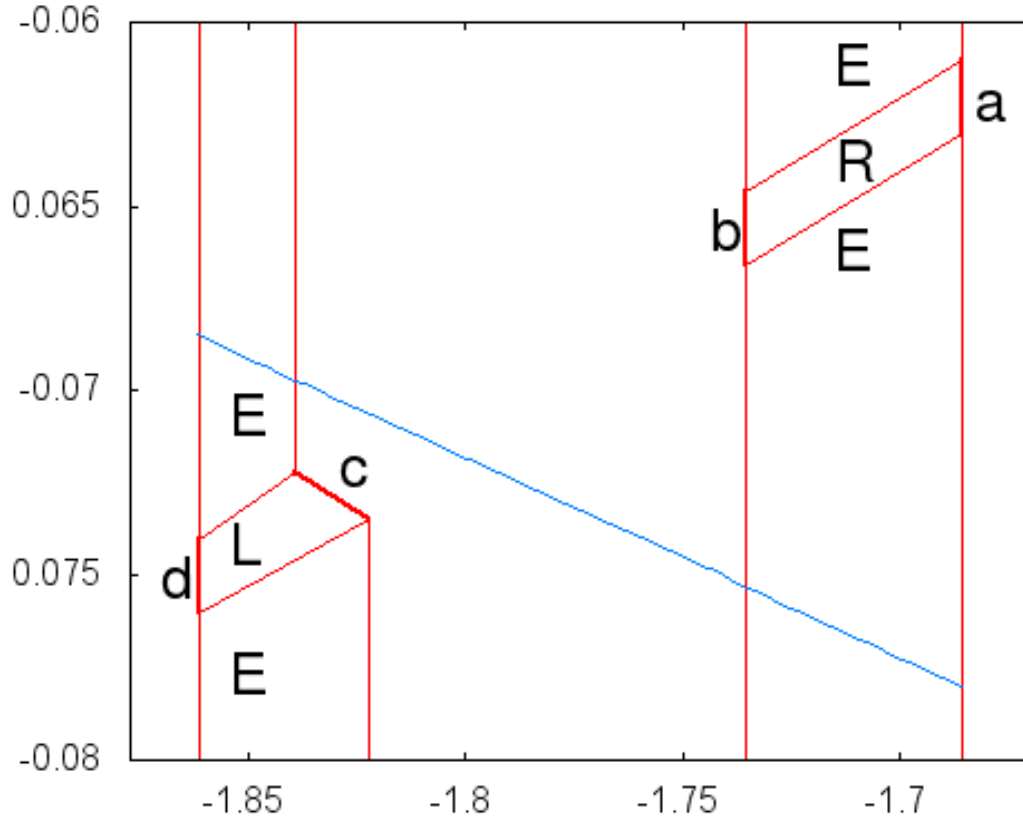


1. Ábra: A (T) Tételben szereplő halmazok

A (T) Tétel topológiai jellegű. Érvényességéhez egyáltalán nem szükséges, hogy az L és az R halmazok téglalapok, az E halmaz pedig négy darab

"egy-egy félegyenes mentén végtelen téglalap" uniója legyen. Egyedül a teljes elrendezés topológiai "minősége" számít, a szakaszok és a félegyenesek nyugodtan meggörbülhetnek.

(K) **Következmény:** Legyen $h : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ a teljes síknak a teljes síkra történő kölcsönösen egyértelmű, inverzével együtt folytonos leképezése, azaz homeomorfizmusa. Ekkor a (T) Tétel érvényes marad, ha a benne szereplő $E, O_C, R, O_R, L, O_L, a, b, c, d$ halmazokat rendre a $h(E), h(O_C), h(R), h(O_R), h(L), h(O_L), h(a), h(b), h(c), h(d)$ halmazokkal, a $\varphi : L \cup R \rightarrow \mathbb{R}^2$ leképezést pedig a $h\varphi h^{-1} : h(L \cup R) \rightarrow \mathbb{R}^2$ leképezéssel pótoljuk.



2. Ábra: A (K) Következményben szereplő halmazok, amikor azt a (3) Vallis modellre alkalmazzuk.

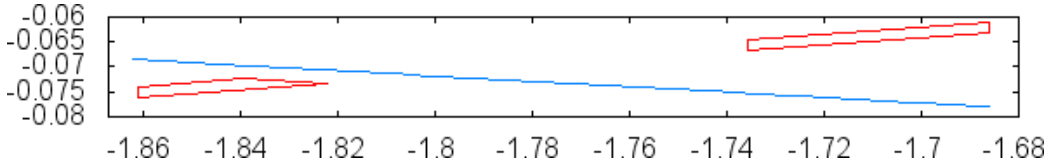
Az egyes négyszögek csúcspontjainak koordinátái a jobb felső csúcspontról

kezdve az óramutató járásával ellentétes irányban:

L: (-1.839,-0.07218), (-1.861,-0.074), (-1.861,-0.076), (-1.822,-0.07346)

R: (-1.686,-0.061), (-1.7356,-0.0646), (-1.7356, -0.0666), (-1.686, -0.063)

Az L és az R négyszögek csúcspontjai számítógépes kísérletezéssel adódnak. Meghatározásuk a számítógépes káosz-bizonyítás legfontosabb lépése.



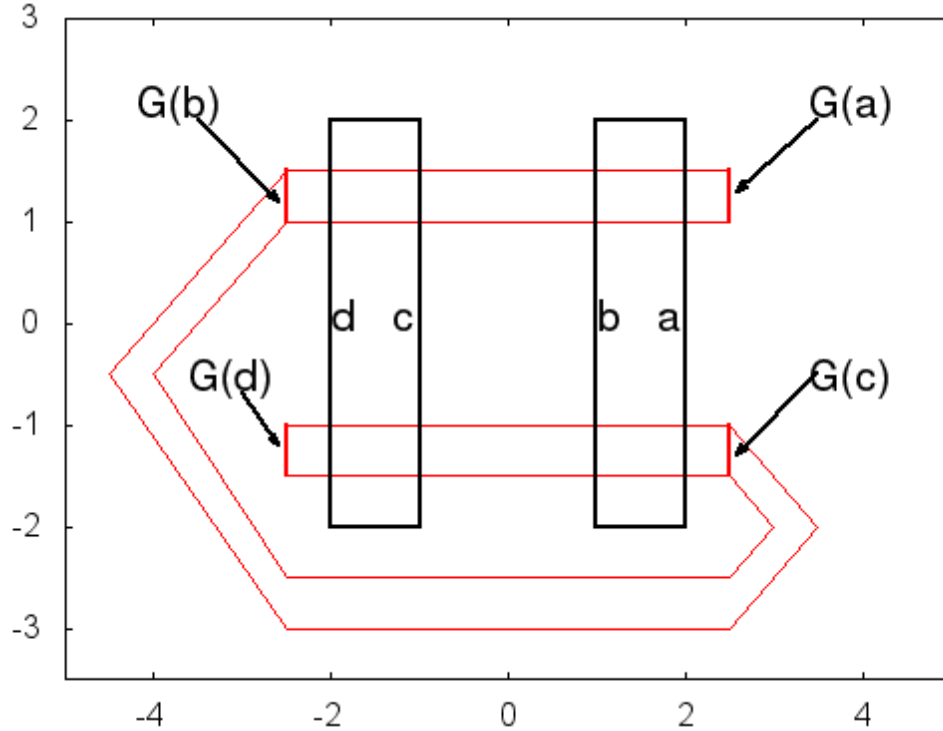
3. Ábra: Az $L \cup R$ arányos képe

A (K) Következmény valamint a vele rokon állítások egész családja alapvető szerepet játszik a számítógéppel segített káosz-bizonyításokban. A tipikus alkalmazásokban az $X = \mathbb{R}^2$ sík egy háromdimenziós autonóm közönséges differenciálegyenlet-rendszer Poincaré metszősíkja, $h\phi h^{-1}$ pedig a Poincaré visszatérési függvény valahányadik iteráltja. Magát a h homeomorfizmust külön nem szokták megadni, sőt az ábrákon is — az ábra önmagáért beszél — $h\phi h^{-1}$ helyett φ és $h(R), h(L), h(E), \dots$ helyett maga $R, L, E, \dots$ szerepel. A számunkra fontos alkalmazáshoz az $\mathbb{R}^3$ tér $z = 0.3$ egyenletű síkja és rajta az $R, L, E, \dots$ halmazoknak a 2. és 3. Ábrán megadott elrendezése tartozik.

A 4. Ábrán Smale stilizált G-patkó-leképezése látható. A káosz szempontjából csak az álló L és R téglalapok, valamint az ezek affin képeiként adódó felső (top) $T = G(R)$ és alsó (bottom) $B = G(L)$ fekvő téglalapok a lényegesek. (A stilizált G-patkó G leképezése értelmelve van az ábra két fekete téglalap közötti részén is: Ez utóbbi tartomány képhalmaza a stilizált G-patkó $G(b)$ és $G(c)$ függőleges szakaszok közötti része, amely a B és a T téglalapokat stilizált G betűvé egészíti ki.)

A stilizált G-patkó-leképezés eleget tesz a (K) Következmény feltételeinek s mint ilyen, modellként szolgál a TDK dolgozatomban bemutatott alkalmazáshoz. Smale sokkal ismertebb U-patkó-leképezése az L és az R téglalapok egyikén megőrzi, másikán megváltoztatja a körüljárást. Ez magyarázza, hogy a háromdimenziós tér autonóm közönséges differenciálegyenleteinek Poincaré féle visszatérési függvényeivel kapcsolatban nem az U-patkó, hanem a G-

patkó a szokásos objektum. A G-patkó felfedezése egyébként — pontosan ezért — néhány héttel megelőzte az U-patkó felfedezését.



4. Ábra: A szokásos G-patkó egy megjelenítése

1.3. Az *El Niño* Vallis modellje³

TDK dolgozatomban célja topologikus Smale patkó létezésének bizonyítása az

$$\left. \begin{aligned} \dot{x} &= By - Cx \\ \dot{y} &= xz - y \\ \dot{z} &= -xy - z + 1 \end{aligned} \right\} \quad [20], \text{ cf. (11) } [21] \quad (3)$$

differenciálegyenlet-rendszerben — a szokásos paraméterértékek $B = 102$, $C = 3$ —, amelyet Vallis (1986) [20] vezetett be az egyenlítői Csendes óceánban megfigyelhető hőmérsékleti fluktuációk leírására. Maga a természeti

³a (3) rendszer a Vallis [20] és mások [22] által definiált számos modell közül a legegyszerűbb

jelenség — amelyet perui halászok már a XVIII. században *El Niño*-nak⁴ neveztek el (Vallis az *El Niño*-Déli Oszcilláció megnevezést használja) — elképesztően bonyolult. Vallis (3) modellje a valóság nagyfokú leegyszerűsítése⁵, és mégsem kevés az, amit megmutat belőle. Az x változó az oda-vissza kelet–nyugati irányúnak tételezett felszíni óceánáramlás sebessége, az y és a z pedig speciális hőmérsékleti változók. A z változó a felszíni vizek átlaghőmérséklete, az y változó pedig a nyugati és a keleti óceáni medence felszíni vízhőmérséklet-különbségének fele. Képlettel

$$z = \frac{T_w + T_e}{2} \quad , \quad y = \frac{T_w - T_e}{2} \quad ,$$

ahol tehát T_w és T_e az Észak–Ausztrália illetve Peru partjai előtti vizek hőmérséklete. A (3) differenciálegyenlet-rendszer egydimenziós-kétpontos modell, amely a felső vízréteg a szél hatását is figyelembe vevő advekciós egyenletének diszkretizációjából származik.

A TDK dolgozatban vizsgált (3) modell a $p = 0$ speciális esete az ugyan-csak Vallis által két évvel később (1988) [21] bevezetett egyparaméteres

$$\left. \begin{aligned} \dot{x} &= By - C(x + p) \\ \dot{y} &= xz - y \\ \dot{z} &= -xy - z + 1 \end{aligned} \right\} \quad (4)$$

modellcsaládnak.

1.4. A Lorenz és a Vallis rendszer összehasonlítása

A számítógépes feladatok nagyobb részének elvégzése után vette észre témavezetőm a (3) Vallis rendszer és az (1) Lorenz rendszer közötti matematikai kapcsolatot: a Vallis rendszert az

$$x = -X \quad , \quad y = -\frac{C}{B} Y \quad , \quad z = -\frac{C}{B} Z + 1 \quad (5)$$

transzformáció az

$$\left. \begin{aligned} \dot{X} &= C(Y - X) \\ \dot{Y} &= \frac{B}{C} X - Y - XZ \\ \dot{Z} &= XY - Z \end{aligned} \right\} \quad (6)$$

⁴azaz magyarul "Fiúgyermek", tekintettel arra, hogy évente (de nem minden évben) karácsony előtt jelentkezik a parti vizek szokatlan felmelegedésének formájában, amelyet az ünnep után heves viharok és hatalmas esőzések követnek.

⁵annyira egyszerű, hogy Vallis (2006) [22] monográfiájában már említés sem történik róla.

egyenletrendszerbe viszi, amely nem más, mint a Lorenz rendszer, a $\sigma = C$, $b = 1$, $r = \frac{B}{C}$ választással. A $B = 102$, $C = 3$ paraméterértékeknek speciálisan a $\sigma = 3$, $b = 1$, $r = 34$ paraméterértékek felelnek meg.

Különösen érdekes tehát, hogy a meteorológus Lorenz (1963) [9] és az óceanográfus Vallis (1986) [20], [21] végső soron ugyanazt a közönséges differenciálegyenlet-rendszert vezette le, jóllehet a két rendszer a változók fizikai jelentése alapján egymással nem hasonlítható össze⁶. A konkrét fizikai tartalom a paraméterek és változók számértékeiben már sem az (1), sem a (3) egyenletrendszerben sem jelenik meg: A matematikai szakirodalomban mindegyikük transzformált-dimenziótlanított formában kerül tárgyalásra.

A (5) transzformáció régóta ismert kell legyen, de semmi esetre sem köztudott. A (3) és az (1) differenciálegyenlet-rendszer numerikus viselkedése közötti hasonlóságokra többen is felfigyeltek, Vallistól [20], [21] kezdve egészen a vonatkozó bifurkáció-sorozatok összehasonlítását is elvégző Magnitskii–Sidorov szerzőpárosig [10], [11]. Mivel a $\sigma = 10$, $b = \frac{8}{3}$, $r = 28$ és a $(B = 102, C = 3 \Leftrightarrow \sigma = 3, b = 1, r = 34)$ paraméterhármas számértékei egymástól markánsan különböznek, az (1) és a (3) (azaz a (6)) rendszerek numerikus viselkedése között nemcsak hasonlóságok, hanem érdemi különbségek is megfigyelhetők.

A (3) és az (1) rendszerek ekvivalenciája szerepel Tung (2007) [19] könyvében, aki a (4) feladatot az eddigi $p = 0$ esetén kívül a $p = 0.83$ paraméter-választás mellett is tárgyalja⁷.

TDK dolgozatomban folytatásaként a (4) rendszer kaotikus voltát a p paraméter Tung [19] által kijelölt értéke esetén is szeretném igazolni, s amennyiben ez sikerül, az eredményt egy *Chaos in Vallis' asymmetric Lorenz model for El Niño* című cikkben megjelentetni.

⁶Lorenz (1) modellje egy Fourier-sorfejtés csonkolásából származik, x - y - z változóinak jelentése pedig rendre a konvektív intenzitás, a felszálló és a leszálló légáramlatok hőmérséklet-különbsége, valamint a függőleges hőmérséklet-profil átlagos eltérése a lineáristól. Figyelemreméltó, hogy ezekhez képest a (3) Vallis rendszer változóinak jelentése (áramlási sebesség, illetve vízhőmérsékletek) mennyire kézzelfogható és közérthető.

⁷a google a "Vallis AND Lorenz" kérdésre többek között a [19] könyv tartalomjegyzékét is kiadta. Három hét múlva maga a könyv is — egy svédországi könyvtár példánya — megérkezett.

2. A SZÁMÍTÓGÉPES FELADAT

A TDK dolgozat a (K) Következményt a (3) egyenletrendszerben alkalmazza.

2.1. Számítógéppel segített káosz–bizonyítás

Egy számítógéppel segített káosz–bizonyítás két összetevőből áll:

- Elméleti rész: a kaotikusság egy, a (K) Következményhez hasonló feltételrendszere
- Számítástechnikai rész: amely önmaga is két feladatra bomlik
 - a feltételrendszerben szereplő halmazok heurisztikát⁸ is igénylő megkeresése az adott dinamikában, ami egy konkrét matematikai sejtéshez vezet
 - majd a sejtés igazolása a feltételek egyenkénti ellenőrzésével: ez intervallum–aritmetikával szigorúvá tett numerikus módszerek segítségével történik

A numerikus módszerek "szigorú" (rigorous) jelzője szakkifejezés, amely az intervallum–aritmetikai számítások legfontosabb jellemzőjére utal: a számítógép automatikusan elvégzi az elvileg lehetséges maximális összhiba elemzését is. Szokásos a "megbízható" (reliable) és a "validált"⁹ (validated) jelzők azonos értelmű használata is.

A tényleges alkalmazás φ leképezését Vallis *El Niño* modelljének a $z = 0.3$ sík alulról felfelé történő átmetszéseihez tartozó Π Poincaré visszatérési függvényének második iteráltja szolgáltatja. A Π valamint a $\varphi = \Pi \circ \Pi$ második iterált létezését az $L \cup R$ halmaz egy kis környezetében a szigorú numerikus módszerekből származó számítógépes tapasztalat biztosítja. Az

⁸A szakirodalomból számos kísérlet ismert ezeknek a heurisztikus eljárásoknak az automatizálására. A teljes siker aligha valószínű [3], mert ehhez egy nagyméretű globális constraint satisfaction feladatot kellene megoldani.

⁹A validált szónak az intervallum–aritmetikában saját értelme van, amely különbözik a szó általánosan érvényes "helyességbizonyítás" jelentésétől. Itt jegyezzük meg, hogy egy numerikus módszer szigorúvá tétele általában a gépidő harmincszoros–hatvanszoros megnövekedésével jár.

előkészítő nem-szigorú (esetünkben MATLAB) heurisztika soha nem vezethet el matematikai bizonyossághoz, hanem csak sejtéseket eredményezhet. A φ folytonossága az átmetszések transzverzalizálásának — hiszen a $\Pi(L \cup R)$ és a $(\Pi \circ \Pi)(L \cup R) = \varphi(L \cup R)$ halmazok egy kis környezetében $\dot{z} = -xy - z + 1 > 0$, szintén a szigorú számítógépes tapasztalat alapján — következménye. (A $\dot{z} > 0$ egyenlőtlenség azt is mutatja, hogy az átmetszések — ellentétben a Lorenz rendszerben megszokott iránnyal¹⁰ — alulról felfelé történnek.)

Az egyes trajektóriáknak a $z = 0.3$ síkkal való metszéspontját azonban nem lehet pontosan kiszámolni. Amire a VNODE¹¹ képes, az az, hogy meghatározzon két kicsiny méretű téglatestet (háromdimenziós intervallumot), amelyek tartalmazzák a valódi, pontos trajektória egy-egy, a tényleges átmetszéshez közeli pontját. Az első téglatest teljes egészében a $z = 0.3$ sík alatt, a második téglatest teljes egészében a $z = 0.3$ sík felett helyezkedik el.¹² Az alulról felfelé történő átmetszéseknek megfelelően mindig a második téglatest az időben későbbi. A hozzájuk tartozó időpontok különbsége az *átmetszési időintervallum*, amely a VNODE néhány egymás utáni lépésközének összege. (A teljes egészében a $z = 0.3$ sík alatt elhelyezkedő *első*, valamint a teljes egészében a $z = 0.3$ sík felett elhelyezkedő *második* téglatest között ugyanis további, a $z = 0.3$ síkot metsző téglatestek is lehetségesek.) A térbeli közelség időbeli közelséget is jelent, amely elvben tetszőlegesen kicsivé tehető. A gyakorlatban addig kell a közelítő téglatesteket továbbra is $z = 0.3$ sík alatt illetve felett tartva változtatni, amíg a köztük lévő térbeli és időbeli távolság olyan kicsi nem lesz, hogy segítségükkel a tényleges átmetszési pont első és második koordinátájára elegendően éles becslés legyen adható. Az elegendően éles becslés a (T) Tételben szereplő feltételeknek a $z = 0.3$

¹⁰amint az a Z változó bevezetésének (5) képletéből is kitűnik

¹¹közönséges differenciálegyenletek kezdetiérték-feladatait az intervallum-aritmetika segítségével numerikusan megoldó programcsomag

¹²a túlságosan nagy téglatesteket eleve elkerülendő, a VNODE számára előírjuk, hogy az első téglatest a $z = 0.299$ és a $z = 0.3$, a második téglatest a $z = 0.3$ és a $z = 0.301$ síkok között legyen. A $0.001 = 0.301 - 0.3 = 0.3 - 0.299$ állandó a numerikus számológépek általunk, a számítógépes kísérletezés eredményeképpen választott bemenő paramétere. (Amikor már sejteni lehetett, hogy a gépidő mindvégig elviselhető marad, a 0.001 értéket a legutolsó, a tényleges bizonyítást jelentő futtatás előtt a tizedére csökkentettem. Az alternatíva az lett volna, hogy az a posteriori becslésben törekedjem a maximális élességre.) A VNODE adaptív lépésközüválasztással működik. A maximális lépésköz megválasztására, jóllehet azt a tolerancia-beállításokkal befolyásolni tudjuk, a VNODE nem biztosít közvetlen lehetőséget.

síkon történő ellenőrzését jelenti. Ez az ellenőrzés az intervallum–aritmetika sajátosságai szerint a gépi számítások kerekítési hibákat is tartalmazó összhibájának ún. "worst–case" analízisét is magában foglalja. Ezért számít matematikai erejű és érvényességű bizonyításnak.

2.2. A célzott találgatások heurisztikája

A (3) Vallis rendszer attraktorának metszete a $z = 0.3$ síkkal két enyhén ívelt "holdacska", amelyek hasonlósága a klasszikus Lorenz rendszer holdacskáihoz ([17], [6]) már az első pillantásra is szembeötlő. A holdacskák lokálisan intervallumok és Cantor–halmazok szorzatai; ebből a Cantor–halmazos struktúrából az általam használt maximális számítógépi felbontás finomsága két párhuzamos gyöngysort enged megmutatni. Az 5. Ábra a $P^* = (1, 2, 0.3)$ pontból induló trajektória és a $z = 0.3$ egyenletű sík néhány ezer egymást követő metszéspontját, pontosabban a

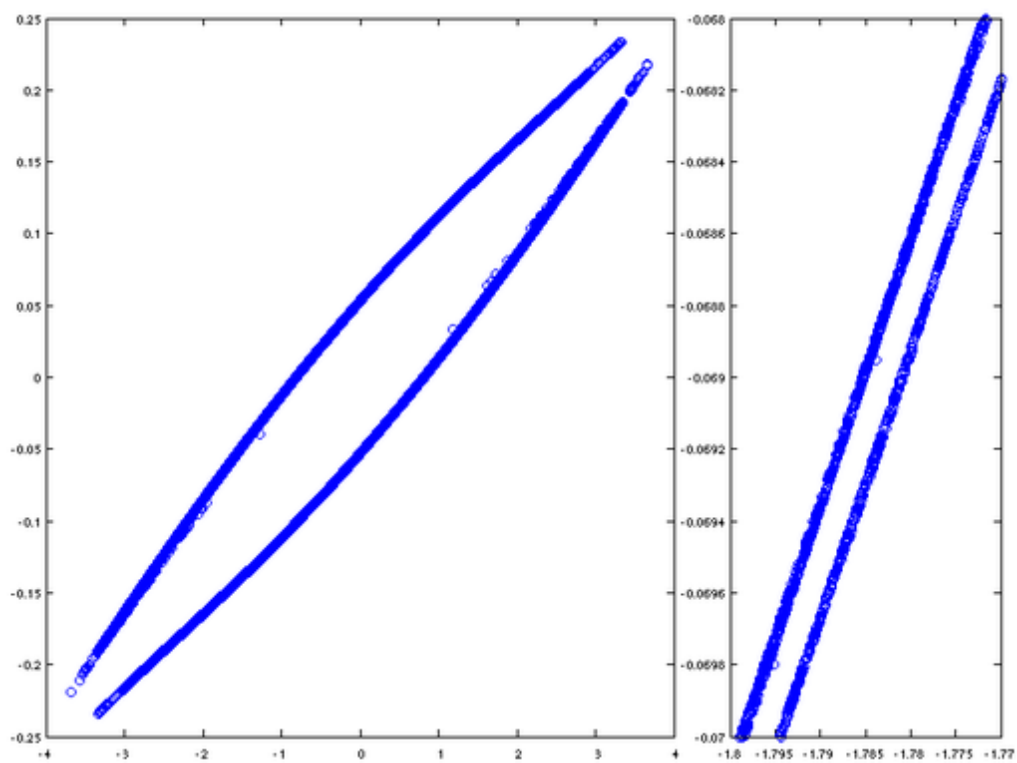
$$\varphi^k(P^*) = \{(x_k(P^*), y_k(P^*))\}_{k=100}^{100000}$$

sorozat elemeit mutatja. Mivel az 5. Ábra holdacskáinak mindegyike szigorúan monoton növekvő folytonos függvény grafikonjának tűnik, logikus arra gondolni¹³, hogy az attraktor felső holdacskájának belső dinamikájáról az

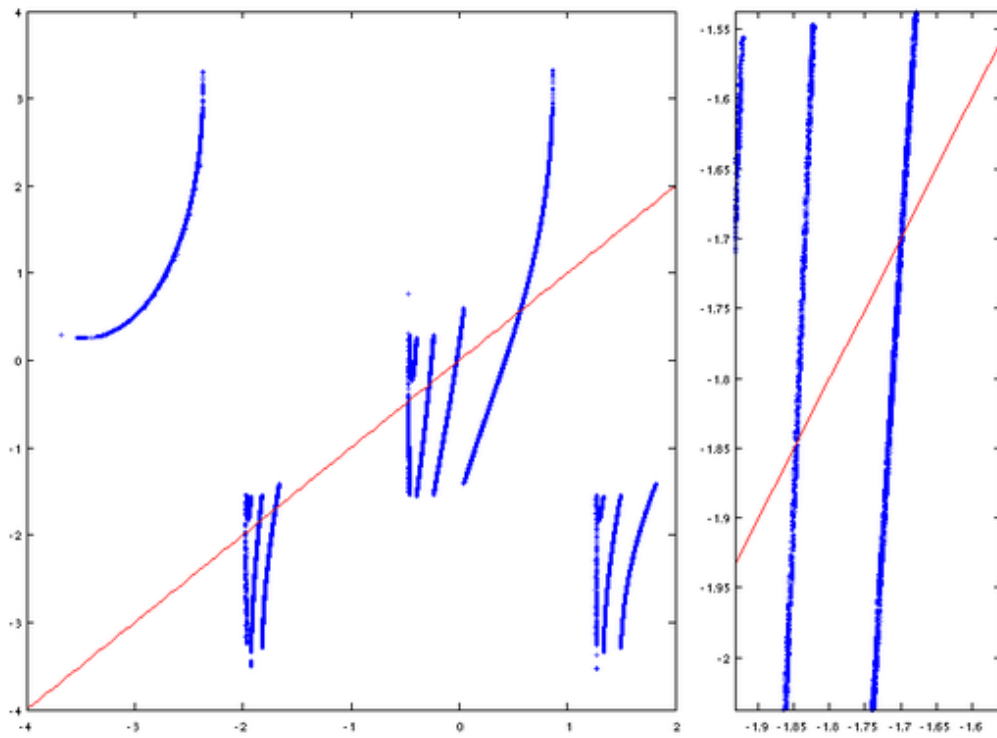
$$\mathcal{P} = \mathcal{P}_{pearl}^{upper} : x_k(P^*) \rightarrow x_{k+1}(P^*)$$

"gyöngysor"–leképezés lényegi információkat tartalmaz. A $\mathcal{P}$ leképezés azokra az $x_k(P^*)$ valós számokra van értelmezve, amelyekre az $(x_k(P^*), y_k(P^*))$ és az $(x_{k+1}(P^*), y_{k+1}(P^*))$ pontok (a $k \in \mathbb{N}$ indexekre a $100 \leq k \leq 100000$ egyenlőtlenség érvényes) mindegyike a felső holdacskán helyezkedik el. Ezt a véges halmazon definiált leképezést azonban úgy tekintjük, mintha néhány nyílt intervallum unióján értelmezett folytonos függvény volna. A 6. Ábrán bemutatott véges ponthalmazzal a számítógépes kísérletezés szempontjából is úgy bánunk, mintha egy valós függvény grafikonjával állnánk szembe és bátran alkalmazzuk rá Smale (T) Tételének egydimenziós vátozatát.

¹³ez a megfontolás már Lorenz [9] eredeti cikkében is szerepel



5. Ábra: A Vallis rendszer holdacskái, valamint a felső holdacska egy *erősen kinagyított* részlete.



6. Ábra: A $\mathcal{P}$ gyöngysor-leképezés grafikonja, valamint e grafikon az ℓ és az r intervallumokhoz vezető, *mérsékelten kinagyított* részlete.

(A) **Állítás:** Legyenek ℓ és r diszjunkt, korlátos és zárt intervallumok. Legyen továbbá $\phi : \ell \cup r \rightarrow \mathbb{R}$ folytonos függvény és tegyük fel, hogy

$$\ell \cup r \subset \phi(\ell) \quad \text{és} \quad \ell \cup r \subset \phi(r) \quad (\text{cf. 2}). \quad (7)$$

Ekkor a ϕ leképezés az $\ell \cup r$ halmazon $\ell - r$ értelemben kaotikus.

Ha az ℓ és r halmazok megkeresésének amúgy egyáltalán nem nehéz feladatát a 6. Ábrán kísérletezve megoldottuk, akkor a (K) Következmény alkalmazhatóságához szükséges L és R "téglalapokat" az 5. Ábra felső holdacska mentén vett parallelogrammák¹⁴ formájában kereshetjük, a tabu halmazokat kijelölő irányt pedig függőlegesnek vehetjük.

Mindez természetesen nem más mint heurisztika, de olyan heurisztika, amelynek érdemes számítógépes kísérletekkel utánamenni.

A (7) tulajdonság egyszerű következménye, hogy a φ leképezésnek mind az ℓ , mind az r intervallumban létezik egy-egy, összességében tehát legalább két fixpontja.

Magától értetődő, hogy a $\mathcal{P}$ "gyöngysor"-leképezést $\varphi = \Pi \circ \Pi$ helyett definiálhattuk volna magára a Poincaré féle Π visszatérési függvényre. Ekkor azonban — vizsgáljuk akár a felső, akár az alsó holdacska belső dinamikáját¹⁵ — a vele "azonosított" valós függvény fixpontmentes volna, és így az (A) Állítást nem tudnánk alkalmazni. Ez magyarázza, hogy numerikusan a Π helyett elég hamar már annak második iteráltjával kezdtem foglalkozni.

A 6. Ábrán egyébként a heurisztikus-hipotetikus fixpontok száma 7; ennyiszer metszi a gyöngysor az $y = x$ egyenest. Kézenfekvő az

$$\ell = [-1.86, -1.82] \quad , \quad r = [-1.73, -1.68]$$

választás¹⁶.

A számítógépes kísérleteket hosszabb-rövidebb MATLAB, GSL és VNODE programok egymásutánjával végeztem el. A tényleges bizonyítás VNODE

¹⁴vagy legalábbis parallelogrammákhoz közeli (a felső holdacska-hoz illesztett lokális koordináta-rendszerben $\ell \times [-\varepsilon, \varepsilon]$ illetve $r \times [-\varepsilon, \varepsilon]$, $0 < \varepsilon \ll 1$ alakú) topologikus négyszögek

¹⁵az (1) Lorenz rendszer $x \rightarrow -x$, $y \rightarrow -y$, $z \rightarrow z$ szimmetriája miatt ez amúgy is teljesen mindegy

¹⁶ez a választás az (A) Állításnak teljesen megfelel, a (K) Következmény alkalmazhatóságához azonban még sok munka van hátra.

programjának mintegy kétnapos futtatása előtt sokszáz "trial and error" numerikus kísérlet elvégzésére volt szükség. A bevezető próbálkozásokat jellemzően MATLAB, az ezeket finomító kísérleteket többnyire GSL, a befejezést közvetlenül előkészítő számításokat pedig VNODE környezetben végeztem el. A kísérletek száma az előkészítő, a geometriát nagy vonalakban feltérképező MATLAB fázisban volt a legnagyobb. A MATLAB saját grafikus kijelzőjének interaktivitása ezt a munkát komoly mértékben tudta felgyorsítani. A VNODE által kezelt adatok hatalmas mennyisége miatt nehézkes az ábrák gyors készítése: Nem egy esetben szükség volt az adattömbökön belüli közvetlen kutakodásra.

Mindezek így, együtt vezettek el a végső soron sikeresnek bizonyult L és R négyszögek megtalálásához. A 2. és a 3. Ábrán látható kék, csaknem egyenes vonal a φ leképezés szakadási pontjainak halmaza az $L \cup R$ halmaz egy kis környezetében. Ez az elméletileg és numerikusan is roppant veszélyes vonal (amely az origó mint egyensúlyi helyzet stabil sokaságának [17], [6] lokális metszéspontja a $z = 0.3$ síkkal) miatt nem lehetett az L négyszög c oldalát függőlegesre választani. Sokat próbálkoztam a c oldal balra mozgatásával, de ez minden esetben a (K) Következmény egyik vagy másik feltételének szinte azonnali megsértéséhez vezetett. Könnyebb dolgom volt a b és a d oldalakkal: mindkettőjüket úgy sikerült függőlegesnek választanom, hogy közben a négyszögek maguk minél kisebbek legyenek¹⁷.

Az alfejezet lezárásaként az általam használt három számítógép technikai paramétereit ismertetem, illetve megadom a MATLAB, a GSL, és a VNODE egyfajta összehasonlítását.

Az általam használt számítógépek:

- Intel Atom; 1.6GHz, egymagos laptop
- Intel Pentium 4; 2.4GHz, egymagos asztali gép
- Intel Core i7; 3.4GHz, nyolcmagos asztali gép

¹⁷ami a (2) feltétel $\varphi(L) \cup \varphi(R) \subset X \setminus E$ részének VNODE általi ellenőrzésére fordítandó gépidő csökkentése miatt volt fontos: a (2) feltétel "egydimenziós", $\varphi(a) \cup \varphi(c) \subset O_R$ és $\varphi(b) \cup \varphi(d) \subset O_L$ részének ellenőrzése ehhez képest összehasonlíthatatlanul rövidebb időt vesz igénybe.

A MATLAB, a GSL, és a VNODE sebességét úgy hasonlítottam össze, hogy mindegyikkel ugyanazt a számítási feladatot oldottam meg. Ez a feladat a $z = 0.3$ síkhoz tartozó Poincaré követőfüggvény második iteráltjának kiszámítása volt éspedig egy s ugyanazon szakasz 1001 pontjában.¹⁸ A megfelelő kezdetiérték-problémák megoldására az egyes programrendszerek beépített algoritmusait használtam, default toleranciákkal.

Az Intel Pentium 4 számítógépen elvégzett összehasonlítás fő adatait az alábbi Táblázat tartalmazza.

PROGRAMCSOMAG	ALGORITMUS	FUTÁSI IDŐ	SZIGORÚ?
MATLAB (JAVA)	ODE45 $\Leftrightarrow$	198sec	nem
GSL (C++)	$\Leftrightarrow$ RKF	0.24sec	nem
VNODE (C++)	HO	122sec	igen

Az ODE45 jelölés két egymásba ágyazott, egyenként negyed- és ötödrendű Runge–Kutta módszer Fehlbergtől származó kombinációjára utal ("leánykori nevén" az RKF algoritmusra), míg a HO rövidítés feloldása Hermite–Obreshkov. Ez utóbbi egyike a Taylor-módszerek [1] családjának. Fontos megemlíteni, hogy az intervallumos világban egészen más a differenciálegyenlet-megoldó módszerek hasznossági hierarchiája, mint a hagyományos, nem-intervallumos világban. A Taylor típusú módszerekről éppen csak annyit tanultam a Numerikus Módszerek tantárgyban, hogy elméletileg fontosak ugyan, de (legalábbis a nem-intervallumos világban) soha nem szabad használni egyiküket sem.

¹⁸A tényleges feladat a $\{\varphi(S_k)\}_{k=0}^{1000}$ pontsorozat meghatározása volt, ahol S_k a 2. Ábrán látható L négyszög bal oldalát képező függőleges d szakasz ezer egyforma részre történő felosztásának k -adik osztópontja. Képlettel

$$S_k = (-1.861, -0.063) + k(0.0, 0.000002), \quad k = 0, 1, \dots, 1000.$$

Ez egy tipikus részfeladat, amelyen a MATLAB, a GSL és a VNODE specialitásai, előnyei és hátrányai egyaránt jól szemléltethetők. (A GSL esetén 100000 osztópontot vettem és az eredményt visszaosztottam 100-zal.)

2.3. Egy a posteriori becslés

A kiindulás az $\dot{\mathbf{r}} = \mathbf{f}(\mathbf{r})$ vektoros alakba átírt (3) rendszer megoldó-operátora lineáris közelítésének

$$\begin{aligned}\Phi(t, \mathbf{r}) - \mathbf{r} - t\mathbf{f}(\mathbf{r}) &= \int_0^1 (1-s)\Phi''_{tt}(st, \mathbf{r}) ds \cdot t^2 \\ &= \int_0^1 (1-s)[\mathbf{f}'(\Phi(st, \mathbf{r}))](\mathbf{f}(\Phi(st, \mathbf{r}))) ds \cdot t^2\end{aligned}\quad (8)$$

Jacobi mátrixos, integrálmaradéktagos változata. Az integrandus értéke a $B = 102$, $C = 3$ paraméterválasztás mellett az

$$(1-s) \begin{pmatrix} -3(102y - 3x) + 102(xz - y) \\ z(102y - 3x) + (-1)(xz - y) + x(-xy - z + 1) \\ -y(102y - 3x) + (-x)(xz - y) + (-1)(-xy - z + 1) \end{pmatrix} \quad (9)$$

vektor. Az (8) képlettel rokon

$$\Phi(t, \mathbf{r}) - \mathbf{r} = \int_0^t \mathbf{f}(\Phi(s, \mathbf{r})) ds = \int_0^t (\mathbf{f}(\Phi(s, \mathbf{r})) - \mathbf{f}(\mathbf{r}) + \mathbf{f}(\mathbf{r})) ds$$

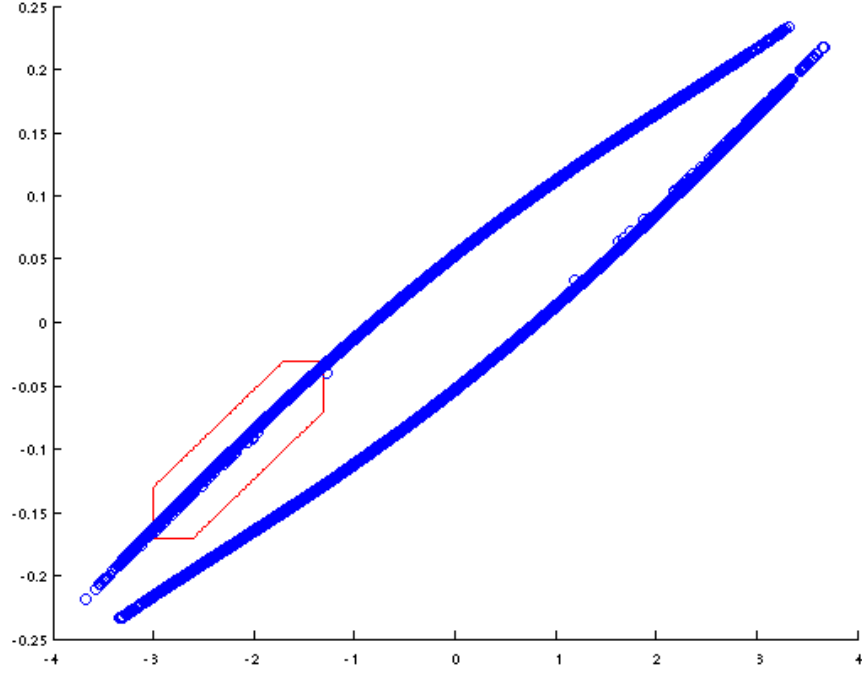
Newton–Leibniz formula normabecslése a

$$|\Phi(t, \mathbf{r}) - \mathbf{r}| \leq L \int_0^t |\Phi(s, \mathbf{r}) - \mathbf{r}| ds + Mt,$$

majd ebből a Gronwall Lemma a

$$|\Phi(t, \mathbf{r}) - \mathbf{r}| \leq Mt + \frac{M}{L}(e^{Lt} - 1 - Lt), \quad t \geq 0 \quad (10)$$

egyenlőtlenséghez vezet. Itt az M és az L értéke az $\mathbf{f}(\mathbf{r})$ vektor euklideszi normájának illetve az $[\mathbf{f}'(\mathbf{r})]$ Jacobi mátrix normájának maximuma a 7. Ábrán látható $z = 0.3$ sík $\varphi(L \cup R)$ halmazának egy $H \times [0.299, 0.3]$ alakú féloldali környezetében, ahol a H hatszöget később specifikáljuk.



7. Ábra: A Vallis rendszer holdacskái a befoglaló H hatszöggel

A VNODE program egy hívásának

- általunk biztosított inputja
a $z = 0.3$ sík egy T téglalapja, a *tárgy-téglalap*,
- számunkra a későbbiekben szükséges outputja
egy $\tau = \tau(T) > 0$ számérték, az *átmetszési időintervallum*, és
az $\mathbb{R}^3$ tér egy $K = K(T)$ téglatestje, a *kép-téglatest*.

Itt a T téglalap az $L \cup R$ halmazt elegendően finoman lefedő $\{T_k\}$ téglalap-család tetszőleges elemét jelenti, a K téglatest pontjainak z koordinátái pedig eleget tesznek a $0.299 < z < 0.3$ egyenlőtlenségnek.

Amire ténylegesen szükségünk van, az a $\varphi(T)$ halmaz, amelyről csak annyit tudunk, hogy

$$\varphi(T) \subset \text{proj}\Phi([0, \tau], K) ,$$

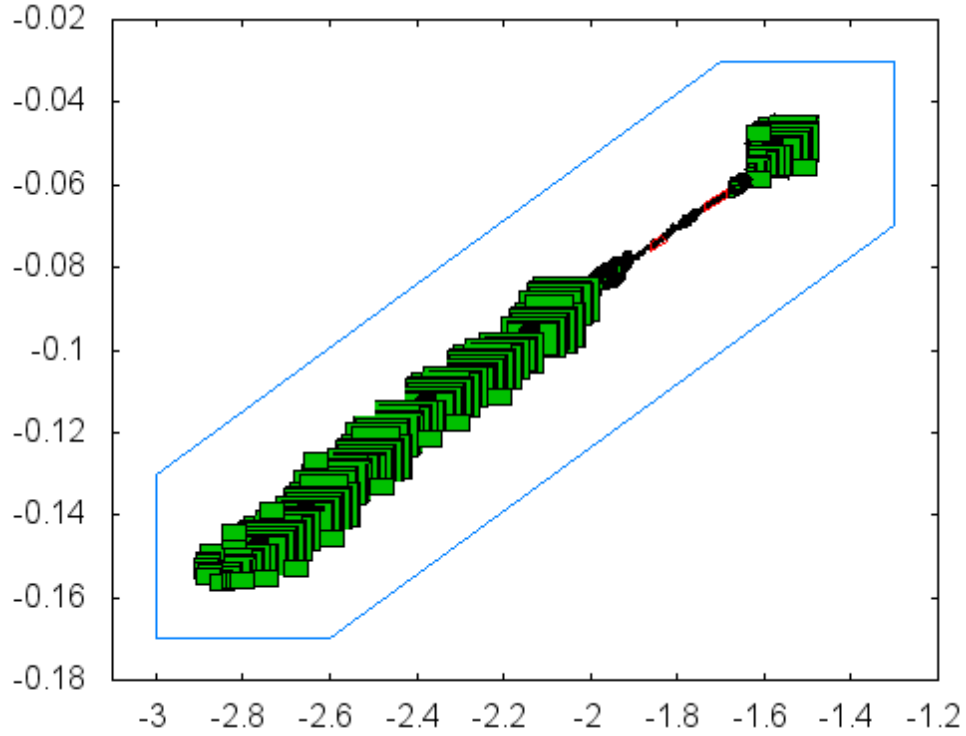
ahol Φ a (3) Vallis rendszer megoldó-operátora,

$$\text{proj} : \mathbb{R}^3 \rightarrow \mathbb{R}^3, (x, y, z) \rightarrow (x, y, 0.3)$$

pedig a $z = 0.3$ síkra történő merőleges vetítés. A T választásától függően a (2) feltétel a $\varphi(T_k) \subset O_R$, $\varphi(T_k) \subset O_L$, $\varphi(T_k) \subset X \setminus E$ előírások egyikét jelenti. Elegendő tehát megmutatnunk, hogy

(C_0) a $\varphi(T_k)$ halmaz, illetve az őt tartalmazó $\text{proj}\Phi([0, \tau], K_k)$ halmaz nem metsz bele a (2) feltétel által kijelölt tabu halmazba.

A legnagyobb nehézség az, hogy a $\{T_k\}$ téglalapcsaládot menet közben kell meghatározni.



8. Ábra: Kép-téglatestek egy halmazának vetülete a $z = 0.3$ síkon, a befoglaló H hatszöggel együtt. A kiindulási tárgy-téglalapok az $L \cup R$ halmaz egy fedését alkotják.

A $\text{proj}\Phi([0, \tau], K_k)$ halmaz becslésére az (8), (9), (10) képleteket hívhatjuk segítségül.¹⁹ Az előzetes numerikus tapasztalatok azt mutatják, hogy

$$\tau = \max_k \tau_k < 0.0003 \quad \text{és} \quad \cup_k \text{proj}K_k \subset H, \quad (11)$$

ahol

$$H = \text{conv}\{P_1, P_2, P_3, P_4, P_5, P_6\}, \quad \text{a}$$

$$P_1(-1.7, -0.03, 0.3), P_2(-3.0, -0.13, 0.3), P_3(-3.0, -0.17, 0.3), \\ P_4(-2.6, -0.17, 0.3), P_5(-1.3, -0.07, 0.3), P_6(-1.3, -0.03, 0.3)$$

pontok által kijelölt konvex hatszög. A H hatszög kiválasztása viszonylag durva közelítés, hiszen $\cup_k \text{proj}K_k$ közel van a H hatszög belsejében elhelyezkedő, $(-1.5, -0.05, 0.3)$ és $(-2.8, -0.15, 0.3)$ végpontú szakaszhoz. A 8. Ábrának megfelelően²⁰ azt tapasztaljuk, hogy

$$\text{az } \cup_k \text{proj}K_k \text{ és a } \partial H \text{ halmazok távolsága} > 0.004, \quad (12)$$

ahol ∂H a H hatszög területét/peremét jelenti. A H hatszög fenti választása mellett M és L értékre nagyvonalú felső becsléssel $M = 11$ és $L = 105$ adódik. A $t = \tau = 0.0003$ választással a (10) egyenlőtlenség jobb oldala kisebb mint a 0.004 számérték, ami geometriailag azt is jelenti, hogy az

$$\cup_k \varphi(T_k) \subset \cup_k \text{proj}\Phi([0, \tau_k], K_k) \subset \text{proj}\Phi([0, \tau], \cup_k K_k)$$

halmazok mindegyike is a H hatszögben helyezkedik el.

Továbbra is nagyvonalú becslésekkel számolva azt kapjuk, hogy $(x, y, z) \in H \times [0.299, 0.3]$ és $t \in [0, \tau]$ esetén a (9) formulában szereplő vektor

- első koordinátája a $[-(1-s)130, (1-s)30]$
- második koordinátája a $[-(1-s)8, (1-s)4]$

¹⁹mivel a $\text{proj}\Phi([0, \tau], K_k) \subset X \setminus E$ feltétel időnként a $\text{proj}\Phi([0, \tau], K_k) \subset L \cup R$ alakot ölti és mert az $L \cup R$ halmaz y irányban mindössze 0.002 vastagságú, szükség van a szokásos hibabecslés koordináták szerinti szétbontására. Ehhez a Gronwall Lemma még vektoros változatában sem lenne elegendően éles, csak a τ_k átmetszési lépésköz drasztikus csökkentésével együtt. Így a döntő szerep a továbbiakban az (8)–(9) formuláké. Az a nagy szerencse, hogy a (9) formulában a $t^2 = \tau_k^2$ mellett önmagában is megjelenő $t = \tau_k$ olyan vektorral szorzott alakban szerepel, amelynek a $z = 0.3$ síkra vett merőleges vetülete nagyjából-egészében párhuzamos az L és az R halmazok "iránytangensével".

²⁰a 8. Ábrára még visszatérünk, a 9.-10.-11. Ábrákkal összefüggésben.

intervallumba esik. Mindez lehetőséget teremt tetszőleges $\mathbf{r} \in K_k$ és $t \in [0, \tau_k]$ esetén a $\Phi(t, \mathbf{r})$ és az $\mathbf{r} + t\mathbf{f}(\mathbf{r})$ vektorok különbsége első és második koordinátájának becslésére. (A harmadik, a z koordináta most már lényegtelen, hiszen a (2) feltétel téglalaponkénti ellenőrzésében nem játszik szerepet.) A fentiekből

$$\text{proj}\Phi([0, \tau_k], K_k) \subset \text{proj}K_k + [0, \tau_k] \cdot \text{proj}f(K_k) + ([-65, 15] \times [-4, 2])\tau_k^2 \quad (13)$$

adódik, ahol a jobb oldalon három síkbeli halmaz összege áll. Közülük a középső a $[0, \tau_k] \subset \mathbb{R}$ intervallumból vett számok összességének és a $\text{proj}f(K_k)$ síkbeli halmaznak a szorzata²¹.

A (C_0) célkitűzés helyett tehát elegendő azt ellenőrizni, hogy

(C) a $\text{proj}K_k + [0, \tau_k] \cdot \text{proj}f(K_k) + ([-65, 15] \times [-4, 2])\tau_k^2$ halmaz nem metsz bele a (2) feltétel vonatkozó speciális esete által kijelölt tabu halmazba.

Ez az a feladat, amelyet a $\{T_k\}$ téglalapcsalád megkeresésével együtt teljes egészében a számítógépre bízhatunk. Ha a szigorú számítógépes numerika olyan eredményeket szolgáltat, amelyek a (11) és a (12) feltételek teljesülését is visszaigazolják, akkor készen vagyunk.²²

Valóban ez a helyzet: A számítógépes vizsgálatok adatait a legutolsó, negyedik fejezet tartalmazza.

2.4. Intervallum–aritmetika dióhéjban

A numerikus módszerekben a véges aritmetikai modellek és a megfelelő számábrázolások központi kérdést alkotnak. Mivel a véges számtesten gyakran nem reprezentálhatók az eredmények, ezért a program kerekítésre vagy csonkolásra kényszerül (például eltérő nagyságrendű számok összeadásánál),

²¹jóllehet a szükséges alapokat csak a most következő alfejezet tárgyalja, vegyük észre, hogy az intervallum–aritmetika világában nemcsak K_k , hanem $f(K_k)$ is (háromdimenziós) intervallum és hogy a (13) tartalmazási reláció jobb oldalán szereplő halmazokat az intervallum–aritmetika alpműveletei kapcsolják össze.

²²mivel a VNODE a (C) célkitűzésnek megfelelően kell hogy számoljon mindent, a becslések a posteriori jellege úgy tűnik, két egymás utáni, különböző futtatás elvégzését igényli. A (C) célkitűzést megelőző teljes érvelés szerint azonban egyetlen futtatás is elegendő. (A befejező VNODE futtatás sikeres volta jogossá teszi a H hatszög eredetileg nem szigorú numerika szerint történt megválasztását.)

ami sokszor elrontja az eredményt. Az intervallum-aritmetika előnye az, hogy a pontos megoldást mindig magában foglaló intervallumokkal számol, emiatt nincs a fentiekben leírt veszteség, viszont a kerekítési hibák növekedésével az intervallumok is nőnek. Ezen tulajdonsága miatt, előszeretettel alkalmazzák tartalmazási problémáknál, mint amilyen ez a feladat is. Manapság biztonságos és jól használható eszközt nyújtanak a numerikus módszerek és az analitikus függvények intervallumokon értelmezett megfelelői, mivel nagy és kiterjedt irodalmuk van, például Alefeld és Majer(2000) [1], Zilinskas(2005) [24].

Az alapl műveletek a következőképpen vannak definiálva. Legyen I és J két egydimenziós, korlátos és zárt intervallum.

$$I \oplus J := \{x \oplus y | x \in I, y \in J\}$$

$$[a, b] + [c, d] = [a + c, b + d]$$

$$[a, b] - [c, d] = [a - d, b - c]$$

$$[a, b] * [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$$

$$[a, b] / [c, d] = [\min(a/c, a/d, b/c, b/d), \max(a/c, a/d, b/c, b/d)]$$

(A 0-t tartalmazó intervallummal való osztás nem értelmezett.)

Az összeadás és a szorzás művelet asszociatív és kommutatív, de nem disztributív, hanem szubdisztributív (annak megfelelően, hogy a kerekítési hiba nem független a műveletek sorrendjétől): Az $I(J + K)$ halmaz részhalmaza az $IJ + IK$ halmaznak.

Az intervallumok egymáshoz képest lehetnek:

- Kisebbek vagy nagyobbak
- Átlapolóak (alul, felül és két irányból is)

Ezért a rendezési relációk definíciója nehézségekbe ütközik.

Az intervallumok megadása kétféleképpen történhet:

- Egy pontos értéket egy kicsiny hosszúságú, nulla középpontú, két végével megadott intervallummal összeadunk:
 $I = 5.0 + \text{interval}(-1,1)1.0\text{E-}10 \text{ // } [5.0-1.0\text{E-}10; 5.0+1.0\text{E-}10]$
- Az intervallum két végének pontos megadásával:
 $I = \text{interval}(123,456) \text{ // } [123;456]$

- Megadható egy pontból álló nulla sugarú intervallum is, ha a két vége azonos:

```
I = interval(10,10) // [10;10]
```

A kiíratásnál segítségemre volt az intervallum középpontját és a sugarát visszaadó két függvény (midpoint és rad). Ezekkel lebegőpontos formátumba lehetett konvertálni a számokat, amelyeket felhasználtam az ábrák rajzolásához, illetve megállapítottam a maximális eltéréseket. Definiáltam két függvényt (LEnd, UEnd), amik az intervallum alsó, illetve felső határát adják vissza lebegőpontos formátumban. E két függvény a kód olvashatóságát nagyban növeli.

Ezek formátuma a következő:

```
interval I
double m,r
I = interval(-5, 5) //[-5; 5]
m = midpoint(I) //0
r = rad(I) //10
sup = UEnd(I) = midpoint(I) + rad(I) //5
inf = LEnd(I) = midpoint(I) - rad(I) //-5
```

Az intervallum geometriailag n -dimenziós téglatestként szemléltethető. A háromdimenziósakra téglatestként, a kétdimenziósakra téglalapként fogok hivatkozni. A sík intervallumos ábrázolásban egy vastagság nélküli téglatestként szemléltethető. Azért téglatest, mert bár a harmadik dimenzióban nincs vastagsága (rad), értéke (midpoint) van. (Akkor lehet vastagsága, ha a harmadik dimenzió nem reprezentálható egy nulla vastagságú intervallummal; ilyen például a $z = \sqrt{2}$ sík. Ez a dolgozatban nem fog előfordulni, csak az általánosság miatt kell majd kezelni ezt az esetet is.)

2.5. A felhasznált eszközök

A VNODE-LP programot N. S. Nedialkov [15] írta. A VNODE-LP megbízható (reliable), szigorú (rigorous) numerikus közönséges differenciálegyenlet-rendszer kezdetiérték probléma megoldó program. Intervallum-arit-

metikával dolgozik és a kezdetiérték probléma megoldásának befoglaló intervallumait számolja ki, ezáltal biztosítván a szigorúságot. A program felhasználja a nagy múltú LAPACK és BLAS lineáris algebrai programcsomagokat [2] és [5], illetve választható hozzá a Profil 2.0 [7] és a FILIB++ [8] intervallum programcsomag. A dolgozatban a Profilt használtam. A VNODE-LP-ben megadtam az egyenletet, a kezdeti és a befejezési időt, így a program az általa választott lépésközzel elvégezte a számítást a lehető legpontosabban. A programban nincs lehetőség közvetlenül befolyásolni a lépésköz maximális nagyságát. Alapvetően nagy lépésközökkel dolgozik, ezért egy logaritmikus kereséshez hasonló eljárással a kezdeti érték idejének és a befejezési időnek a változtatásával sikerült kikényszeríteni a kisebb lépésközöket. Fontos megjegyezni, hogy az időparaméter is intervallum típusú, ezért előfordulhat olyan eset, hogy az idő nem pontos, hanem ún. lebegése van. (Például: lebegőpontosan nem reprezentálható időértéknél.) A program alapállapotban ezt elkerüli és csak igen ritka esetekben lehet rákényszeríteni, viszont azokkal az esetekkel is számolni kell. Minden esetben szigorúan befoglaló intervallum lesz az eredmény, szemmel tartva a véges aritmetika reprezentálhatóságának korlátait, kifelé kerekítve. Az intervallum típuson az alapl műveletek vannak értelmezve, a szokásos jelölésekkel. Viszont a rendezési relációk nincsenek megvalósítva, ezért ezt a hiányosságot is külön kellett kezelni. A beépített (egydimenziós) intervallum típus szép kiírása (pretty-printing) illetve egy intervallum vektor típus (a több dimenziós intervallumok létrehozására) is rendelkezésre állt. Ezek fölé definiáltam a saját intervallum-koordináta típusaimat (*point3* illetve *point3_fs*) valamint ezek műveleteit²³. A program írása során az olvashatóságot és a logika követhetőségét elsődleges szempontnak tekintettem. Ezeket az alapl működéshez szükséges feladatokat, külön-külön függvényekkel valósítottam meg és így is használtam a későbbiekben. Az ábrák elkészítéséhez a Gnuplot [23] programot használtam. A programom által generált adatokat fájlba mentettem a megfelelő formátumban: az intervallumok középpontjával és két végpontjával megadva mind a függőleges és a vízszintes koordináták szerint. Ezeket az adatfájlokat shell scriptekkel dolgoztam fel és segítségükkel generáltam az ábrákat is. Az ábrák csak a szemléltetést szolgálják. Bizonyító jelleggel nem bírnak. A program Linux környezetben GCC fordítóval C++ nyelven készült. A fentiektől eltérő, egyéb

²³ezekre a kód VNODE-től való nagyobb függetlensége miatt volt szükség. Mert így a pontok is helyettesíthetők más, például lebegőpontos megvalósítással. (Lásd: a GSL-val végzett kísérletek.)

különleges függősége nincs.

A kísérleteknél felhasználtam a GNU Scientific Library 1.15 (továbbiakban GSL) numerikus integráló képességét. Ahogy a mérések is mutatják a táblázatban, sokkal gyorsabban működött a Poincaré metszéspontok kiszámításában. A VNODE működését akartam lemásolni, hogy minél több kódot újra tudjak hasznosítani a már elkészült (VNODE alapú) programból. Így a GSL alacsonyszintű függvényeit használtam, amikhez a Runge-Kutta-Fehlberg-45 módszert választottam a rendelkezésre álló léptető függvények közül. Automatikus lépésközzel történő integrálást határoztam meg, az abszolút hiba 10^{-6} -ra állításával. Ezeket a függvényeket egy, VNODE interfészével meg egyező osztályba zártam. Így a meglévő kódok újrahasznosíthatóvá váltak és nem kellett őket újra megírni. Ezután tudtam méréseket végezni a sebességről, amik eredményeként nagyságrendekkel több ponton tudtam rövidebb idő alatt kísérleteket végezni. Ennek köszönhetően lényegesen több ponttal sokkal pontosabb képet kaptam a dinamikáról, illetve a szakadási-helyekről. Ezek a számítások nem voltak alkalmasak a bizonyításra, csak a gyors kísérletezést és a rendszer megismerését szolgálták. A bizonyító erejű programban csak az így kapott közelítő eredmények pontos ellenőrzését kellett elvégezni. A kísérletező (GSL alapú) program érdekesebb részei közösek a bizonyító (VNODE alapú) programmal. Így külön nem tárgyalom a kódot. Minden a dolgozattal kapcsolatos kód publikálásra kerül és megtalálható lesz a <https://github.com/dlazesz> oldalon.

2.6. A program felépítése vázlatosan

Funkcionális stílusban írtam a programot. Minden elvégzendő feladat egy-egy függvényben van definiálva, illetve a konstansok is ezért vannak globálisan megadva. Kiindulva egy befoglaló intervallumból, a program Branch and Bound módszerrel felosztja az intervallumot (szükség szerint), illetve egy specifikált módon és ellenőrzi a megfelelő feltételeket, hogy szükséges-e további felosztás. A Branch and Bound függvény kimenete a következő: a kezdeti intervallum jónak való felosztása STL vektor típusban (intervallum formátumú elemekkel), amivel további számításokat lehet végezni. Az általam definiált *iterate* függvény a bemenő adatokból kiszámolja a differenciálegyenlet az előre definiált $z = 0.3$ síkra vett Poincaré visszatérési függ-

vényének második iteráltját a globálisan beállított paraméterek²⁴ alapján. Ezáltal egy szigorúan sík alatti és egy szigorúan sík feletti téglatestet adva eredményül. Ezen a függvényen belül történik egy logaritmikus kereséshez hasonló eljárás (*logsearch*), ami biztosítja hogy a téglatestek a síkhoz legközelebb eső nem átölelő intervallumok legyenek. Ehhez a *solve* megoldófüggvény alkalmazása szükséges. Egy, a logikát tükröző, reprezentációtól elválasztott feltétellel vizsgáljuk meg a leképezendő, illetve az eredményként kapott téglalapokat is, külön-külön. Ez a függvény mintegy harminc egyszerű, logikai értékű függvényből tevődik össze, amik a konstansként megadott pontokból intervallum típusban felírt egyenesek egyenleteiből, és egymás kompozícióiból állnak össze. Ez a rész különösen bonyolult, mert az összes kombinatorikusan előforduló topológiai lehetőséget jól kell kezelni benne. A programban felhasználók általam definiált speciális típusokat, amik az adatabsztrakcióval segítik a program logikai struktúrájának megértését. Ezen kívül még vannak segédfüggvények, amik a típusok közötti konverziót végzik. A program több "szépen kiírató" (*pretty-printing*) függvényt is használ. Mivel a hibák több logikai szinten fordulhatnak elő, a programban vannak fordításkor állítható kapcsolók, amik a bőbeszédűséget (*verbosity*) a feladathoz igazítják. Fontos megjegyezni, hogy törekedtem a program átlátható szerkezetére és univerzális felhasználhatóságára. Viszont a feladat jellegéből adódóan a geometriai sajátosságokra kellett mindenhol támaszkodni, ezért a közvetlen újrafelhasználhatóság limitált²⁵, ugyanakkor a probléma megoldása során szerzett tapasztalatok sok helyen programkód szinten is segíthetik az újrafelhasználást²⁶.

2.7. A program részei

2.7.1. A Branch and Bound módszer

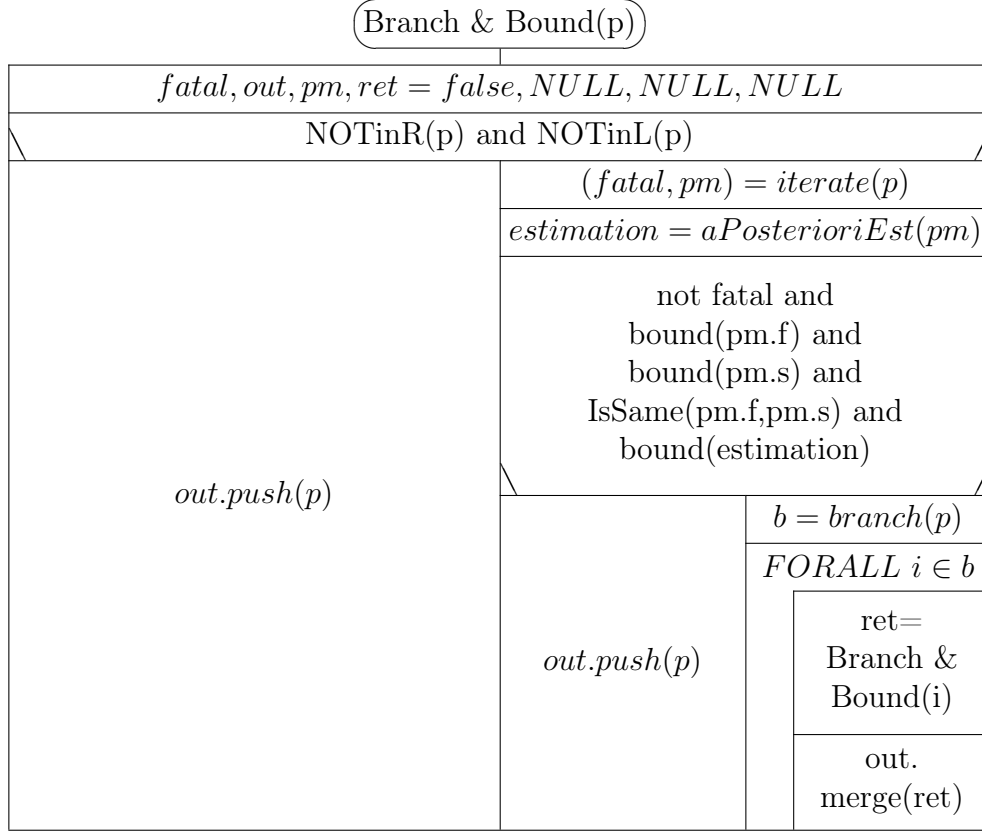
Néhány eltérés tapasztalható a klasszikus Branch and Boundtól ebben az

²⁴ezen paraméterek segítségével a függvény teljesen univerzális tud maradni. A konkrét paramétereket a későbbiekben tárgyalom.

²⁵a négyszögek definícióját és a hozzájuk tartozó logikai függvényeket illetve a metszősíkra és átmetszésére vonatkozó feltételeket kell csak átírni. Viszont a mennyiségük miatt a program nem elhanyagolható részét teszik ki.

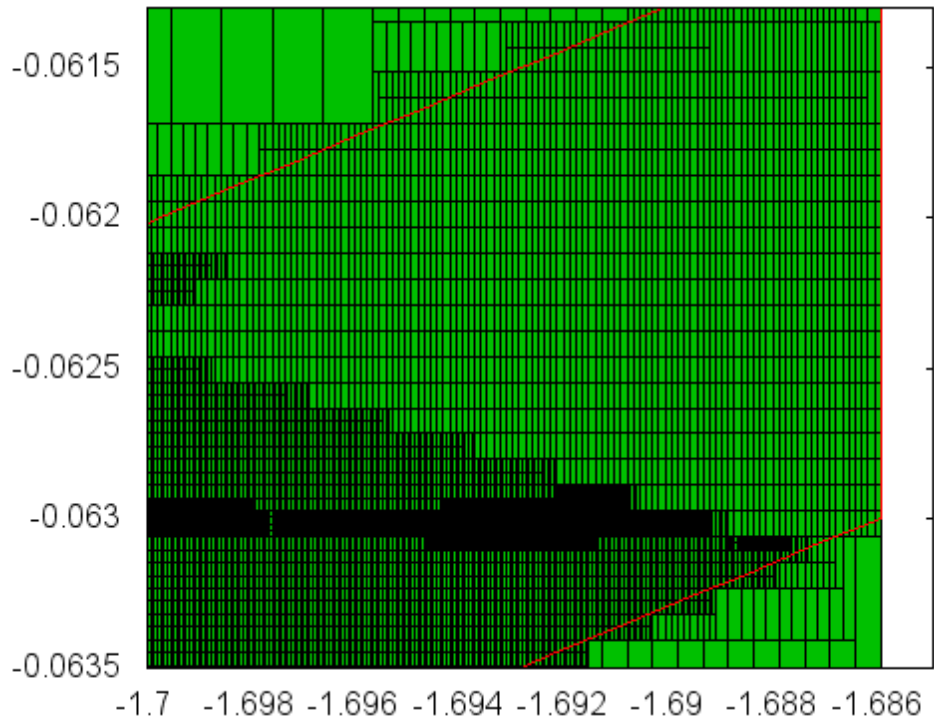
²⁶A kód újrafelhasználásra kerül a közeljövőben, amikor a klasszikus Lorenz rendszerbeli káoszbizonyítást ellenőrizzük, illetve a (4) $p=0.83$ esetében a harmadik iterációban fogunk káoszt keresni.

implementációban:



Elsőként szembetűnő az, hogy mivel befoglaló intervallumról van szó, feltétlenül elfogadja az intervallumot, és kilép, amennyiben a paraméter intervallum diszjunkt az $L \cup R$ halmaztól, mert a kezdő intervallum ezen részeit nem akarjuk leképezni sem. Amennyiben van metszés az $L \cup R$ halmazzal kiszámolja a bemenő paraméter intervallumnak, a szintén paraméterként kapott, jelen esetben a $z = 0.3$ síkhoz tartozó Poincaré-féle visszatérési függvény (jelen esetben második) iteráltját. Ez a művelet lehet sikertelen is, ha nagy intervallumokról van szó, amelyek szakadási helyeket is magukban foglalhatnak. Ekkor nagyon megnő az intervallum szélessége, és egy küszöb után sikertelennek nyilvánítja a számolást a program. Ha a számolás sikeres, visszatér két intervallummal, pontosabban a sík alatti és a sík feletti, a síkhoz legközelebb eső, de a síkot nem metsző téglatest-párral. Ezekből előállítja az elméleti számolás alapján adódó a posteriori befoglaló becslést,

amire végrehajtja a Bound függvényt. Ha a Bound függvény és a második kép számolásának eredménye egyaránt pozitív, akkor elfogadja az intervallumot, egyéb esetben a Branch függvénnyel szétbontja és sorban mindegyik részre rekurzívan meghívja magát, a szokásos Branch and Bound eljárásnak megfelelően.



9. Ábra: Az R négyszög egy részének fedése tárgy-téglalapokkal.

A Bound függvény közvetlenül támaszkodik a logikai függvényekre, amiket külön fogok tárgyalni. A Branch függvénynél egy nagyon egyszerű, minden optimalizációs lehetőséget nélkülöző heurisztika szerint a "hosszabbik oldalt felezi" és így két intervallummal tér vissza. Az intervallumok szigorú felezése nem triviális feladat, mert a paraméter intervallum hosszabbik oldal szerinti felezőpontja eshet egy nem reprezentálható számra, ezért befoglaló intervallumot kell gyártani ebben az esetben is. Külön tárgyalom az eljárás részleteit.

2.7.2. Az intervallumok felezése: A Branch függvény

A hosszabbik oldal kiválasztása után a többi adattag érintetlenül hagyásával, a hosszabbik oldalt reprezentáló intervallum két széléből készít két egyponthoz tartozó intervallumot. Képezi ezeknek az átlagát, ami már nem feltétlenül nulla szélességű intervallum. Ezután az eredeti alsó széléből és az átlag felső széléből képez egy intervallumot, illetve az átlag alsó széléből és az eredeti felső széléből képezi a másikat.

Képletekkel:

$$\begin{aligned} Input &= [a; b] \\ A &= [a; a] \\ B &= [b; b] \\ Atlag &= ([a; a] + [b; b]) / 2 // [c; d] \\ Also &= midpoint(Atlag) - rad(Atlag) // [c; c] \\ Felso &= midpoint(Atlag) + rad(Atlag) // [d; d] \\ KI_1 &= [A; Felso] // [a; d] \\ KI_2 &= [Also; B] // [c; b] \end{aligned}$$

Az így kapott két intervallum nem feltétlenül diszjunkt, viszont az uniójuk megegyezik a kiinduló halmazzal és nem veszünk a kerekítésen semmit. Természetesen, ha egy nulla szélességű intervallumból indul ki ez az eljárás, akkor a végtelenségig képes vágni. Ezért mindenekelőtt van egy ellenőrzés arra, hogy nem egyponthoz tartozó intervallummal van-e dolgunk, mert ha mégis, akkor programhibával ki kell lépni²⁷.

Vegyük észre, hogy a Branch and Bound függvény már csak kétdimenziós intervallumokkal dolgozik, az *iterate* függvény pedig csak háromdimenziós intervallumokkal.

A 9. Ábra a 2.3 alfejezet végén megfogalmazott ($\mathcal{C}$) tulajdonság VNODE általi ellenőrzése közben konstruált $\{T_k\}$ tárgy-téglalapcsalád²⁸ egy, az R négyyszög jobb oldali harmadához tartozó tagjait szemlélteti²⁹.

²⁷A kód újrafelhasználása során észrevettem, hogy ez a megoldás sem teljesen véd az egyenesek rossz beállítása ellen. Elég hosszú ideig tart, mire hibával megáll.

²⁸az ábrán szereplő téglalapok egy "menet közbeni" állapotot mutatnak be és további felbontásra várnak. A Branch and Bound eljárás átlagos, 16-18 körüli veremmelysége a printer felbontóképességénél sokkal kisebb méretű téglalapokhoz vezet.

²⁹A számolások végeztével, megállapítottuk, hogy szélétében sokkal érzékenyebb tud lenni

2.7.3. A Poincaré visszatérési függvény második iteráltjának képzése

Azt vizsgáljuk, hogy a $z = 0.3$ egyenletű sík pontjaiból indulva a trajektoriák mikor és hogyan metszik újból ezt a síkot. Tapasztalati tény, hogy 10 időegység alatt megtörténik minden esetben a második alulról történő metszés³⁰. Az időkonstans a konkrét feladattól függő paraméter. A számítás leáll, ha a globális hiba (*GlobalExcess*) értéke meghaladja az 1000 értéket³¹. Ez a konstans is feladatfüggő beállítás, de általánosan is jól használhatónak tűnik. A VNODE megoldóját mindenképpen újra kell inicializálni minden meghívásakor, különben a számolás nem lesz optimális. A metszés, intervallumokról lévén szó, különböző módokon történhet. A program az aktuálisan kiszámolt illetve az utolsó számolási lépés eredményével dolgozik. A z koordináta szerint vizsgálva a már kiszámolt intervallumok sorozatát, olyan téglalapt-párt tekint átmetszésnek, amelynek első tagja szigorú értelemben alatta van $z = 0.3$ síknak, a második tagja pedig szigorú értelemben fölötte³². Az így definiált intervallum-pár alsó és felső közelítését adja a tényleges átmetszési pontnak. A kiválasztott intervallum-pár tagjai az eredeti intervallumsorozatnak nem feltétlenül szomszédos elemei. A második intervallum-pár, ennek megfelelően, a második alulról történő átmetszés alsó-felső közelítése. A logaritmikus keresés is, ez utóbbi intervallum-párt fogja finomítani³³.

a dinamika. Ezért célszerűbb lehet a négyszögek hosszában, hosszúkás, lapos intervallumokra vágni.

³⁰ha nem így történne, hibával azonnal megáll a program és utána be lehet állítani a maximális idő értéket.

³¹Ez a VNODE belső hibaindíkátora. Az 1000 tapasztalati érték kb. a kezdeti intervallumszélesség ezerszeresére való növekedésével ekvivalens. Dokumentáció hiányában nem lehet biztosat állítani. Viszont sokkalta szigorúbb feltételek döntenek az intervallum elfogadásánál, így ez csak egy gyenge feltétel.

³²Ez a viselkedés paraméterként állítható. A klasszikus Lorenz rendszer esetében például a metszősíkot pont fordítva, fölülről lefelé dőfi át a trajektória. Ezért "időben első és második intervallummal" dolgozik a függvény, az "alsó és felső közelítés" megfogalmazás helyett.

³³Az elfogadáshoz szükséges feltétel, hogy a közelítő intervallum-pár síktól legtávolabbi pontja (mindkét esetben) egy bizonyos küszöb alatt legyen. Ezt segíti a logaritmikus keresés.

2.7.4. A logaritmikus keresés eljárás részletei

A logaritmikus keresés a már megtalált, teljesen a $z = 0.3$ sík felett illetve alatt lévő téglatesteken kívül az intervallumok egy harmadik fajtájával is dolgozik. Ez a harmadik fajtájú intervallum olyan téglatest, amelyet a $z = 0.3$ sík elmetesz. Ennek megfelelően a logaritmikus keresés három lépésből áll:

- a) Egy harmadik fajtájú intervallum megtalálása:

A program addig fut, amíg a frissen kiszámolt téglatest és a $z = 0.3$ sík metszete különbözik az üres halmaztól. Minden lépésben kiszámolja az alsó és a felső közelítő téglatestek idejének átlagához tartozó téglatestet. Ha ez utóbbi téglatest és a $z = 0.3$ sík diszjunktak, akkor a megfelelő téglatestet frissíti, mert sikerült megjavítania az egyiket. Különben kilép a ciklusból.

- b) Az alsó téglatest közelítése a trajektória és a $z = 0.3$ sík tényleges metszéspontjához:

A program itt azt vizsgálja, hogy az első lépésben talált metsző téglatest és az alsó téglatest közötti időben tud-e jobb alsó téglatestet konstruálni az időintervallumok felezésével. Addig fut, amíg az idő felezésével olyan intervallumhoz nem jut, amelyik vagy metszi már a $z = 0.3$ síkot, vagy már nincs közelebb a $z = 0.3$ síkhoz mint a korábbi alsó téglatest. Minden körben frissíti az alsó téglatestet, hiszen azt ténylegesen meg tudta javítani függőleges irányban.

- c) A felső téglatest közelítése a trajektória és a $z = 0.3$ sík tényleges metszéspontjához:

Hasonlóan az előző ponthoz, csak ezúttal ellenkező irányból közelít a $z = 0.3$ sík felé.

A számolások ezen részéhez a program a *solve* függvényt használja, ami eltérően az *iterate* függvénytől egy előre meghatározott befejező ideig számol. Vegyük észre, hogy maga a metódus nem egyezik meg a klasszikus logaritmikus kereséssel, viszont a felhasznált elv ugyanaz. Erről kapta a nevét.

2.7.5. A logikai függvények részletei

Az L és az R négyszögek pontjaiból felírjuk az egyenesek egyenleteit. Mindezt intervallumos formában. A meredekségek olyan számok, amelyeket nem lehet véges aritmetikával reprezentálni. Így kiszámolásuknál oda kell figyelni arra, hogy intervallumok hányadosaként legyenek megadva, hogy elkerüljük a kerekítés általi hibákat. Fontos, hogy ezeket az adatokat csak egyszer számolja ki a program az indításkor, mivel nem fognak változni a futás során és az ismételt kiszámolások lassítanák a program futását. Ezért konstansként vannak létrehozva egy olyan struktúrába ágyazva, ami tükrözi az összefüggéseket. A logikai függvények célja, hogy olyan topológiai információkat tudjunk ellenőrizni, mint hogy "Mi az elhelyezkedése az adott pontnak az egyenesekhez képest?". A függvények nevei is ezt a konvenciót követik: Az első betű l,r,u,l rendre left, right, upper, lower utána OF utána l,r,t,b rendre left,right, top, bottom utána L vagy R attól függően hogy melyik négyszögre vonatkozik és végül x vagy y attól függően hogy a síkban vett vízszintes vagy függőleges irányról beszélünk. (Például: lOFrRx: A bal oldalán van-e a pont az R jobb oldalának (x szerint)?) Az ettől eltérő neveket külön tárgyalom. A függvényekben a megfelelő egyenesek egyenletei vannak felírva intervallumos formában, úgy hogy amit lehetett előre kiszámolva letároltunk. A paraméterként adott pont koordinátáit behelyettesítjük az egyenes nullára rendezett egyenletébe és eldöntjük, hogy az eredmény kisebb vagy nagyobb-e mint nulla. Mindezt természetesen intervallumosan. Oda kell figyelni, hogy amikor a < 0 relációt vizsgáljuk az intervallum nullához legközelebbi felső határát vizsgáljuk, illetve a > 0 relációnál az alsó határát. Képlettel:

$$UEnd(I) < 0$$

$$LEnd(I) > 0$$

Így állapítjuk meg, hogy a pont az egyenest határoló félsíkok melyikén helyezkedik el. A mögöttes logikai függvényeket hívom elemi függvényeknek. Ezekből létrehoztam összetett függvényeket, amik az elemi függvények logikai kompozícióiból állnak, és olyan bonyolult kérdéseket válaszolnak meg, mint például: "Benne van-e a vízszintes tengely szerint egy pont az L és az R négyszögek uniójában?", mivel nem elégséges az a feltétele az x szerinti tar-

talmazásnak, hogy a "jobb oldaltól balra van" az adott pont. Vegyük észre, hogy egy intervallum és egy egyenes kölcsönös viszonya alapvetően kétféle lehet: Metszi-e az egyenes az intervallumot vagy nem. Ha nem metszi, akkor az egyenes által meghatározott felsíkok egyikén helyezkedik el. Ha az egyenes átmetszi az intervallumot (főképp a Branch and Bound eljárás elején, amikor nagyok a képintervallumok), akkor az egyszerű teljes tartalmazási relációk mindegyike hamis. Éppen ezért nem elég vizsgálni egy egyszerű feltétel hamisságát vagy igazságát. Az összetett logikai függvények egy osztálya a NOT előtaggal rendelkező, hasonló nevű függvények. Ők biztosítják azt az intuitív negációt, amire szükségünk lesz. Azaz, hogy biztosan nincs egy pont benne a négyszögben a vízszintes és/vagy a függőleges tengely szerint. Továbbá létrehoztam egy *IsSame* nevű függvényt, ami azt ellenőrzi, hogy az alsó és a felső téglatest vetülete a síkra topológiailag ugyanolyan elhelyezkedésű-e³⁴. Ez nagyon hasznos függvény, szakadási helyek keresésekor. Szakadási helyeknél az alsó és a felső közelítő téglatest nagyon eltávolodik egymástól. A $z = 0.3$ síkhoz való közelséget is figyelembe kellett venni, hogy az alsó és a felső közelítés ne eshessen túl távol egymástól, mert ez esetben az a posteriori becslés lehetetlen lett volna az időkülönbség nagysága miatt. Itt a kísérletezéskor enyhébb értékekkel dolgoztam, de a bizonyítóerejű számolás megkövetelte a pontosságot ami lassította a futást. Ez az ellenőrzés is az *IsSame* függvény része. Vegyük észre, hogy kizárólag topológiai információkra támaszkodik a számolás, ezzel növelve az általánosságot. Meg kell jegyezni, hogy a függvények így is erősen támaszkodnak a geometriai tulajdonságokra. Ilyen például az, hogy az R négyszög parallelogramma, amelynek az "álló" oldalai pontosan függőlegesek, és hogy az L négyszög egyik oldala is függőleges. Ezeken felül az L négyszög másik oldalát az *NOTinLvx* függvényben függőleges befoglalóval közelítem a számolás egyszerűsítése végett. A biztonságosság növelése érdekében egy nem topológiai tulajdonságot is ellenőriz a program: Az alsó-felső közelítő téglatest-pár $z = 0.3$ síktól vett maximális távolságát, amit 0.0001-ben maximalizál — ha ennél nagyobb, nem fogadja el a program a téglatesteket³⁵. A tabuhalmazok illetve az, hogy a pont biztosan nincs benne a tabuhalmazban, ami a Bound függvény által ellenőrizendő tény, könnyen megfogalmazhatóvá válik a *NOTinLT* és a *NOTinRT* függvények segítségével. Erősen támaszkodtam a C++ nyelv lusta kiértékelésére, ezzel a logikai

³⁴Amennyiben ez nem áll fenn, a szigorúbb esetre kell vizsgálni illetve további felosztást kell eszközölni.

³⁵Ezzel a paraméterrel korlátoztam a közelítések között eltelt időt, ami az a posteriori becsléshez szükséges egyik paraméter volt.

formulák kiértékelése lehetőség szerint a leggyorsabb az átláthatóság megtartása mellett. (A feltételek falsifikálás céljával lettek megfogalmazva.) Ezzel a strukturálással igyekeztem a programot újra felhasználhatóvá és átláthatóvá tenni, függetleníteni a logikát a reprezentációtól. Egyúttal a buktatókra felhívva a figyelmet a tesztelést és a hibakeresést is segíteni, szem előtt tartva a performancia kérdését is.

Hivatkozások

- [1] Alefeld, G. & Mayer, G., Interval analysis: Theory and applications, *J. Comp. Appl. Math.*, **121**(2000), 421–464.
- [2] Angerson, E., Bai, Z., Bischof, J., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. & Sorensen, D., LAPACK: A portable linear algebra library for high-performance computers, In: *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, New York, 1990, pp. 2–11.
- [3] Bánhelyi, B., Csendes, T. & Garay, B.M., Optimization and the Miranda approach in detecting horseshoe-type chaos by computer, *Int. J. Bifurc. Chaos* **17**(2007), 735–748.
- [4] Bánhelyi, B., Csendes, T., Garay, B.M. & Hatvani, L., A computer-assisted proof for Σ_3 -chaos in the forced damped pendulum equation, *SIAM J. Appl. Dyn. Syst.* **7**(2008), 843–867.
- [5] Blackford, L.S., Demmel, J., Dongarra J., I. Duff, S., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., Pozo, R., Remington, K. & Whaley, L.C., An updated set of basic linear algebra subprograms (BLAS), *ACM Trans. Math. Soft.*, **28**(2002), 135–151.
- [6] Galias, Z. & Zgliczynski, P., Computer assisted proof of chaos in the Lorenz system, *Physica D*, **115**(1998), 165–188.
- [7] Knüppel, O., PROFIL/BIAS V 2.0., Technical Report, Technische Universität Hamburg–Harburg, 1999.
- [8] Lerch, M., Tischler, G., Wolff von Gudenberg, J., Hofschuster, W & Krämer, W., The interval library filib++ 2.0 – design, features and sample programs, Preprint, Universität Wuppertal, 2001.
- [9] Lorenz, E.N., Deterministic nonperiodic flow, *J. Atmos. Sci.*, **20**(1963), 130–141.
- [10] Magnitskii, N.A. & Sidorov, S.V., *New Methods for Chaotic Dynamics*, World Scientific, Singapore, 2006.

- [11] Magnitskii, N.A. & Sidorov, S.V., Transition to chaos in nonlinear dynamical systems described by ordinary differential equations, *Comp. Math. Modelling*, **18**(2007), 128–147.
- [12] Mischaikow, K. & Mrozek, M., Chaos in the Lorenz equations: a computer-assisted proof, *Bull. Amer. Math. Soc.* **32**(1995), 66–72.
- [13] Mischaikow, K. & Mrozek, M., Chaos in the Lorenz equations: A computer assisted proof. Part II: Details, *Mathematics of Computation*, **67**(1998), 1023–1046.
- [14] Mischaikow, K., Mrozek, M. & Szymczak, A., Chaos in the Lorenz equations: a computer assisted proof. Part III: the classical parameter values, *J. Diff. Eq.* **169**(1998), 17–56.
- [15] Nedialkov, N.S., Implementing a Rigorous ODE Solver through Literate Programming, Technical Report, McMaster University, 2010.
- [16] Smale, S., Hirsch, M.W. & Devaney, R.L., *Differential Equations, Dynamical Systems, and an Introduction to Chaos*, Elsevier, Amsterdam, 2004.
- [17] Sparrow, C., *The Lorenz Equations: Bifurcations, Chaos, and Strange Attractors*, Springer, Berlin, 1982.
- [18] Tucker, W., The Lorenz attractor exists, *C.R. Acad. Sci. Paris Ser. Math.* **328**(1999), 1197–1202.
- [19] Tung, K.K., *Topics in Mathematical Modeling*, Princeton University Press, Princeton, R.I., 2007.
- [20] Vallis, G.K., *El Niño* — a chaotic dynamical system? *Science*, **232**(1986), 243–245.
- [21] Vallis, G.K., Conceptual models of *El Niño*/Southern Oscillation, *J. Geophys. Res.*, **93**(1988), 13979–13991.
- [22] Vallis, G.K., *Atmospheric and Oceanic Fluid Dynamics*, Cambridge University Press, Cambridge, 2006.

- [23] Williams, T. & Kelley, C., Gnuplot: command-driven interactive function plotting program,
<http://www.gnuplot.info/> Elérés: 2011 November 2.
- [24] Zilinskas, J., Comparison of packages for interval arithmetic, *Informatica*, **16**(2005), 145–154.

3. A program forráskódja

```
#include <ostream>
#include <sstream>
#include <string>
#include <fstream>
#include <vector>
#include "vnode.h"

using namespace std;
using namespace vnodelp;

#define VERBOSE_DEBUG false
#define VERBOSE_DEBUG2 false
#define VERBOSE_DEBUG2_5 true
#define VERBOSE_DEBUG2_5_2 false
#define VERBOSE_DEBUG3 true
#define GNUPLOT_IMG true
#define GNUPLOT_OBJ false
#define GNUPLOT_OBJ2 true
#define PRINT2 false
#define GNUPLOT_ESTIMATE true
#define NEAR_C false

struct point3{
    interval t;
    interval x;
    interval y;
    interval z;
};

struct point3_fs{
    point3 f;
    point3 s;
};

// Functions:

void discount(void);
point3 aPosterioriEst(point3_fs pm);
vector<point3> branch(point3 p);
bool bound(point3 p);
vector<point3> BB(point3 p,int l);
bool iterate(point3 yp,point3_fs *pm);
point3_fs logsearch(point3 null,point3 end);
point3 solve(interval tend,point3 p);
void header();
void print(point3 p, double GlobalExcess);
iVector point3ToiVector(point3 p);
point3 iVectorTopoint3(iVector y,interval t);
double LEnd(interval x);
double UEnd(interval x);
string PPrintPoint3(string name, point3 p,bool with_t);
string PPrintGnuPlot(point3 p,bool with_t);
string PPrintPoint3WOZ(string name, point3 p,bool with_t);
```

```

// Tests related to the lines:
bool lOFrinRx(point3 p);
bool rOFlinRx(point3 p);
bool rOFrinRx(point3 p);
bool lOFlinRx(point3 p);
bool inRx(point3 p);
bool NOTinRx(point3 p);
bool lOFtinRy(point3 p);
bool uOFbinRy(point3 p);
bool uOFtinRy(point3 p);
bool lOFbinRy(point3 p);
bool inRy(point3 p);
bool NOTinRy(point3 p);
bool NOTinR(point3 p);
bool NOTinRT(point3 p);
bool lOFrinLx(point3 p);
bool rOFlinLx(point3 p);
bool rOFrinLx(point3 p);
bool lOFlinLx(point3 p);
bool FAROFrinLx(point3 p);
bool lOFtrinLx(point3 p);
bool rOFtrinLx(point3 p);
bool rOFbrinLx(point3 p);
bool NOTinLvx(point3 p);
bool inLx(point3 p);
bool NOTinLx(point3 p);
bool lOFtinLy(point3 p);
bool uOFbinLy(point3 p);
bool uOFtinLy(point3 p);
bool lOFbinLy(point3 p);
bool inLy(point3 p);
bool NOTinLy(point3 p);
bool NOTinL(point3 p);
bool NOTinLT(point3 p);
bool uZ(point3 p);
bool aZ(point3 p);
bool oZ(point3 p);
bool aOFZinz(point3 p);
bool uOFZinz(point3 p);
bool onZinz(point3 p);
bool lsSame(point3 p1, point3 p2);

// Need to set these appropriately for the Poincaré section to work well
bool firstBetterZ(point3 first, point3 second);
bool secondBetterZ(point3 first, point3 second);
bool firstGoodZ(point3 p);
bool secondGoodZ(point3 p);
bool changeinz(point3 first, point3 second);
bool reversalinz(point3 p);
bool passinz(point3 p);

template<typename var_type>
void Chaos( int n, // SIZE is the problem
            var_type* yp, // yp is a pointer to output variables
            const var_type* y, // y is a pointer to input variables
            var_type t, // t is the time variable
            void* param) // param is a pointer to additional

```

```

{
    // parameters that can be passed
    // to this function
    interval a(102.0), b(3.0), c(1.0), p(0.0); // OR p(0.83);

    yp[0] = a*y[1] - b*(y[0]+p);
    yp[1] = y[0]*y[2] - y[1];
    yp[2] = -y[0]*y[1] - y[2]+c;
}

class vnodeEngine{
private:
    AD* ad;

public:
    VNODE* Solver;
    interval tend;
    interval z;
    interval zTol;
    int returns;
    double MaxGlobalExcess;

    vnodeEngine(){ // Declare and init Solver
        ad= new FADBAD_AD(4, Chaos, Chaos);
        Solver= new VNODE(ad);
        Solver->setOrder(50);
        MaxGlobalExcess=1000.0;
        tend=10.0;
        returns=2;
        z=0.3;
        zTol=0.0001;
    }

    ~vnodeEngine(){
        delete ad;
        delete Solver;
    }
} VE;

// The first approximation (in time)
bool firstBetterZ(point3 first, point3 second){
    return UEnd(first.z)<UEnd(second.z);
};

// The second approximation (in time)
bool secondBetterZ(point3 first, point3 second){
    return UEnd(first.z)>UEnd(second.z);
};

// The first approximation (in time)
bool firstGoodZ(point3 p){
    return uOFZinz(p);
};

// The second approximation (in time)
bool secondGoodZ(point3 p){
    return aOFZinz(p);
};

```

```

// The position wrt Z is changed
bool changeinz(point3 first, point3 second){
return uOFZinz(first) && !uOFZinz(second);
};

bool reversalinz(point3 p){
return uOFZinz(p);
};

bool passinz(point3 p){
return aOFZinz(p);
};

//-----

void discount(void){ // Discontinuity searcher

point3_fs last,ret;
point3 test0,test,test2;
bool fatal,first;
int m=1500,n=500; // Pieces in x and y
double dx,dy;

test0.t = 0.0;
test0.z = 0.3;
test0.x = interval(-1.739047,-1.68); // Enclosure of the area to be checked
test0.y = interval(-0.08, -0.0750931);

dx=UEnd(test0.x)-LEnd(test0.x);
dy=UEnd(test0.y)-LEnd(test0.y);

for(int j=0;j<m;j++)
{
test=test0;
test.x=interval(LEnd(test0.x) + (j*dx)/m,LEnd(test0.x) + (j*dx)/m);
first=true;

for(int i=0;i<n;i++)
{
test2=test;
test2.y=interval(LEnd(test0.y) + (i*dy)/n,LEnd(test0.y) + (i*dy)/n);
cout << "(" << j << ", " << i << ")" << endl;
cerr << "(" << j << ", " << i << ")" << endl;
//cout << PPrintPoint3("",test2,false)<<endl;
fatal=iterate(test2,&ret);
if (fatal){ cout << "fatal"<< endl; exit(1);}
if(!first &&
(fabs(midpoint(last.f.x)-midpoint(ret.f.x))>1.0E-2 ||
fabs(midpoint(last.s.x)-midpoint(ret.s.x))>1.0E-2 ||
fabs(midpoint(last.f.y)-midpoint(ret.f.y))>1.0E-2 ||
fabs(midpoint(last.s.y)-midpoint(ret.s.y))>1.0E-2
)
){
cout << PPrintPoint3WOZ("Discontinuity:",test2,false)<<endl;
//cout << PPrintPoint3WOZ("f",ret.f,false)<<endl
// << PPrintPoint3WOZ("s",ret.s,false)<<endl;
}
}
}

```

```

        cout<<"Delta(lastfx):"<<midpoint(last.f.x)-midpoint(ret.f.x)<<endl
        <<"Delta(lastsx):"<<midpoint(last.s.x)-midpoint(ret.s.x)<<endl
        <<"Delta(lastfy):"<<midpoint(last.f.y)-midpoint(ret.f.y)<<endl
        <<"Delta(lastsy):"<<midpoint(last.s.y)-midpoint(ret.s.y)<<endl;
    }
    last=ret;

    first=false;
} //fori
} //forj
} // end scont

//-----

point3 aPosterioriEst(point3 fs pm){
// From the lower and upper approximation we get estimates for
// the projected image on the plane. The estimation itself:
    interval T;
    point3 estimation;

    T=interval(0 , UEnd(pm.s.t) - LEnd(pm.f.t) );
    estimation.x=pm.f.x + T*(102*pm.f.y - 3*pm.f.x);
    estimation.y=pm.f.y + T*( pm.f.x*pm.f.z - pm.f.y);
    estimation.z=pm.f.z;

    if(VERBOSE_DEBUG2_5)
        cout << "aPosterioriEst(Before_growth):"
            << PPrintPoint3("",estimation ,false) <<endl;

    estimation.x+=interval(-65,15)*T*T; // Theoretical computations, relying
    estimation.y+=interval(-4,2)*T*T; // on T. Generously overestimated
    // (Maybe better bounds are needed)
    if(VERBOSE_DEBUG2_5)
        cout << "T: ("<< LEnd(T) << " , "<< UEnd(T) << ") \r" << rad(T) <<endl
            << "aPosterioriEst:"<< PPrintPoint3("",estimation ,false) <<endl;

    return estimation;
}

//-----

bool bound(point3 p){
// If the TABOO set is intersected, then immediately branch
// No idea, if it ever stops... Because there is no depth limit

// Not in the TABOO set of L AND Not in the TABOO set of R
// The working mechanism is unknown. At least for now... ;)
    return (NOTinLT(p) && NOTinRT(p));
}

//-----

vector<point3> branch(point3 p){
    point3 temp;
    vector<point3> out;
    interval nlower, nupper;

```

```

if (rad(p.x) > rad (p.y)) // Note: The priority here affects performance
{
    // Cut by X // If L and R lie parallel to X or Y
    if (rad(p.x)==0) //Not saves us from wrong definition of L and R
    {
        cout << "BRANCH: _FATÁL_ERROR, _INTERVAL_IS_ONLY_ONE_POINT_LONG!"
            << endl;
        exit(1);
    }

    if(VERBOSE_DEBUG2) cout << "NEXT: _Cut_by_X" <<endl;

    // [a;b] → [a;a] és [b;b]
    interval lower( LEnd(p.x), LEnd(p.x));
    interval upper( UEnd(p.x), UEnd(p.x));

    // ([a;a] + [b;b]) /2 = [c;d]
    interval middle = (lower+upper)/2;

    // [a;d] és [c;b]
    // => c in [a;d] & d in [c;b]
    interval nlower (LEnd(p.x), UEnd(middle));
    interval nupper (LEnd(middle), UEnd(p.x));

    temp.x = nlower;
    temp.y = p.y;
    temp.z = p.z;
    temp.t = p.t;
    out.push_back(temp);

    temp.x = nupper;
    temp.y = p.y;
    temp.z = p.z;
    temp.t = p.t;
    out.push_back(temp);
}
else
{
    // Cut by Y
    if (rad(p.y)==0)
    {
        cout <<"BRANCH: _FATÁL_ERROR, _INTERVAL_IS_ONLY_ONE_POINT_LONG!"<<endl;
        exit(1);
    }

    if(VERBOSE_DEBUG2) cout << "NEXT: _Cut_by_Y" <<endl;

    // [a;b] → [a;a] és [b;b]
    interval lower( LEnd(p.y), LEnd(p.y));
    interval upper( UEnd(p.y), UEnd(p.y));

    if(VERBOSE_DEBUG2_5_2)
        cout <<"lower ("<< LEnd(p.y) <<","<< LEnd(p.y)<<") "<<rad(lower)<<endl
            <<"upper ("<< UEnd(p.y) <<","<< UEnd(p.y)<<") "<<rad(upper)<<endl;

    // ([a;a] + [b;b]) /2 = [c;d]
    interval middle = (lower+upper)/2;

```

```

    if(VERBOSE_DEBUG2_5_2)
        cout <<"middle"<< midpoint(middle) <<"_ "<<rad(upper)<<endl;

    // [a;d] && [c;b]
    // => c in [a;d] & d in [c;b]
    nlower=interval (LEnd(p.y ), UEnd(middle));
    nupper=interval (LEnd(middle), UEnd(p.y ));

    if(VERBOSE_DEBUG2_5_2)
        cout << "nlower" << midpoint(nlower) <<"_ "<<rad(nlower ) <<endl
            << "p.y" << midpoint(p.y ) <<"_ "<<rad(p.y ) <<endl
            << "nlower" << LEnd( nlower) <<"_ "<< UEnd(nlower) <<endl
            << "p.y" << LEnd( p.y ) <<"_ "<< UEnd(p.y ) <<endl
            << "L" <<((LEnd(nlower))==LEnd(p.y ))<<endl
            << "U" <<((UEnd(nlower))- (UEnd(p.y )))<<endl;

    if(VERBOSE_DEBUG2)
        cout << "This_was_given_to_Branch:_ " << PPrintPoint3("p",p,false)
            << endl;

    temp.x = p.x;
    temp.y = nlower;
    temp.z = p.z;
    temp.t = p.t;
    out.push_back(temp);

    if(VERBOSE_DEBUG2)
        cout << "NEXT:lower:_ " << PPrintPoint3("temp",temp,false) << endl;

    temp.x = p.x;
    temp.y = nupper;
    temp.z = p.z;
    temp.t = p.t;
    out.push_back(temp);

    if(VERBOSE_DEBUG2)
        cout << "NEXT:_upper:_ " << PPrintPoint3("temp",temp,false) << endl;
}

return out;
}

//-----

// Branch & Bound
vector<point3> BB(point3 p,int l){

    vector<point3> b,out,ret;
    vector<point3>::iterator i,oi;
    point3_fs pm;
    point3 estimation;
    bool fatal=false;

    if(GNUPLOT_OBJ) cerr << "GNUPLOT(OBJECT): " <<PPrintGnuPlot(p,false)<<endl;
    if(VERBOSE_DEBUG2) cout << "NEXT:_if_NOTinR(p)_&&_NOTinL(p)" <<endl;

    // This is the object interval: If the object interval does not

```

```

// intersect L U R, accept instantly
// NEAR_C: If we want to approximate just the sloping side C
// to check the conditions of Smale's theorem. This will instantly
// accept other intervals. In other cases it must be set to false

#if NEAR_C
    if ((NOTinR(p) && NOTinL(p)) || FAROFrinLx(p))
    #else
        if (NOTinR(p) && NOTinL(p))
    #endif
    {
        out.push_back(p); // Accepted, push in the output vector

        if(VERBOSE_DEBUG2)
            cout << "BOUND(ObjectNOTin):" << PPrintPoint3("",p,false) <<endl;
        if(GNUPLOT_OBJ2)
            cerr << "GNUPLOT(OBJECT-NOTin):" << PPrintGnuPlot(p,false) <<endl;
    }
    else
    {
        if(VERBOSE_DEBUG2)
            cout << "NEXT:_else_NOTinR(p)_&&_NOTinL(p)" <<endl;

        pm.f.x=pm.f.y=pm.f.z=pm.s.x=pm.s.y=pm.s.z=0; // Just for pretty output
        fatal=iterate(p,&pm);

        if(VERBOSE_DEBUG2)
            cout << "After_iterate" <<endl;
        if(VERBOSE_DEBUG2_5)
            cout << "imgF:_ " << PPrintPoint3("",pm.f,false) << endl
                << "imgS:_ " << PPrintPoint3("",pm.s,false) << endl;

        // From the lower & upper approximation estimates the image on the plane
        // And use this estimation later
        // If fatal, no need for estimation
        if(!fatal) estimation=aPosterioriEst(pm);
        cout << "bound_conditions(true,"<<(true)<<"): "
            << "_bound(pm.f):" << bound(pm.f)
            << "_bound(pm.s):" << bound(pm.s)
            << "_IsSame(pm.f,pm.s):" << IsSame(pm.f,pm.s)
            << "_bound(estimation):" << bound(estimation) << endl;
        if(!fatal && bound(pm.f) && bound(pm.s) &&
            IsSame(pm.f,pm.s) && bound(estimation))
        {
            out.push_back(p); // Accepted, push in the output vector

            cout << "BOUND(Accepted_Object):" << PPrintPoint3("",p,false) << endl;

            // This is the image interval:
            if(VERBOSE_DEBUG3)
                cout << "NEXT:BOUND(Accepted_Image)F_" << PPrintPoint3("",pm.f,true)
                    << endl
                    << "NEXT:BOUND(Accepted_Image)S_" << PPrintPoint3("",pm.s,true)
                    << endl;

            // GnuPlot data goes to stderr to have clear debug output on stdout
            if(GNUPLOT_IMG)

```

```

    cerr << "GNUPLOT: " << PPrintGnuPlot(pm.f, false) << endl
    << "GNUPLOT: " << PPrintGnuPlot(pm.s, false) << endl
    << "T: " << UEnd(pm.s.t) - LEnd(pm.f.t) << endl
    << "Depth: " << l << endl;

    if(GNUPLOT_OBJ2)
        cerr << "GNUPLOT(OBJECT): " << PPrintGnuPlot(p, false)
        << endl;

    if(GNUPLOT_ESTIMATE)
        cerr << "GNUPLOT(ESTIMATION): " << PPrintGnuPlot(estimation, false)
        << endl;

}
else
{
    if(VERBOSE_DEBUG2)
        cout << "NEXT:_branchF" << PPrintPoint3("", pm.f, false) << endl
        << "NEXT:_branchS" << PPrintPoint3("", pm.s, false) << endl;

    b=branch(p);
    // Foreach
    for(i=b.begin(); i!=b.end(); i++)
    {
        if(VERBOSE_DEBUG3)
            cout << "NEXT:BB_recursively(Object)" << PPrintPoint3("", *i, false)
            << endl;

        ret = BB(*i, l++);
        // Push the accepted ones to the output vector
        for(o=ret.begin(); o!=ret.end(); o++) out.push_back(*o);
    }
}

return out; // Return the produced good branches
}

//-----

bool iterate(point3 yp, point3_fs *pm){

VNODE* Solver=VE.Solver;
interval tend=VE.tend;
interval t; // Initialize later to yp.t

iVector y(3);
point3 pnull, pend;
point3_fs passtrough_coord;
bool change=false;
int current_return=0;

if(VERBOSE_DEBUG2)
    cout << "NEXT:_iterate(this_comes_in)" << PPrintPoint3("", yp, false) << endl;

// Saves the last and the current interval for the logarithmic search
pnull=yp; // Last

```

```

pend=yp;    // Current

// Integrate init
y=point3ToiVector(yp);
t=yp.t;    // tnull

// Integrate
if(VERBOSE_DEBUG2) cout << "NEXT: _iterate(this_comes_in)"
                    <<"t0 (" << LEnd(t) <<" , "<< UEnd(t) <<" )\t"
                    <<"tend (" << LEnd(tend) <<" , "<< UEnd(tend) <<" )"
                    << endl;

Solver->setFirstEntry();
Solver->setHmin(0);
Solver->setOneStep(on);
while(t!=tend)
{
    Solver->integrate(t,y,tend);
    if(Solver->successful() && Solver->getGlobalExcess() <= VE.MaxGlobalExcess)
    {
        if(VERBOSE_DEBUG)
        {
            header();
            print(iVectorTopoint3(y,t), Solver->getGlobalExcess());
        }
    }
    else
    {
        cout << "FATÁL_ERROR(mk)!"
              << midpoint(yp.x) <<"\t" << midpoint(yp.y) << endl;
        return true;
    }

    // Updates the saved points
    pnull=pend;
    pend=iVectorTopoint3(y,t);

    // The position wrt Z is changed, we have the first coordinate
    if(changeinz(pnull,pend))
    {
        if(VERBOSE_DEBUG) cout << "Change!" << endl;
        change=true;
        passthrough_coord.f=pnull;
    }

    // The trajectory turned back. No pass, just intersection. False alarm!
    if(change && reversalinz(pend))
    {
        if(VERBOSE_DEBUG) cout << "False_alarm!" << endl;
        change=false;
    }

    // The trajectory has passed through Z.
    // We've got the upper point AND passthrough
    if(change && passinz(pend))
    {
        if(VERBOSE_DEBUG) cout << "Pass!" << endl;

```

```

        change=false;
        current_return++;
        passtrough_coord.s=pend;
    }

    // The number of Poincaré returns is enough
    if (current_return>=VE.returns)
    {
        if(VERBOSE_DEBUG)
            cout<< "Reached_the_"<<VE.returns<<"th_pass!" << endl
                << "NEXT:_logsearch(this_goes_in)"
                << PPrintPoint3("pnull",passtrough_coord.f,false) << endl
                << "NEXT:_logsearch(this_goes_in)"
                << PPrintPoint3("pend",passtrough_coord.s,false) << endl;

        *pm=logsearch(passtrough_coord.f,passtrough_coord.s);

        // This is different from the usual output for easy reading
        if(VERBOSE_DEBUG){
            cout << "pf\t"; print(pm->f,0);
            cout << "ps\t"; print(pm->s,0);
        }

        break;
    } // If Pass. There is no else!

} // While

if (UEnd(t)>=LEnd(tend)){
    cout << "TERE_WEREN'T_"<<VE.returns<<"_POENCARÉ_RETURNS!" <<endl;
    exit(1);
}
return false; // Not fatal
} // Nth iterate

//-----

// Logarithmic search
point3_fs logsearch(point3 null,point3 end){

point3_fs current;
point3 pm=null,pm2; // Middle interval
point3_fs out; // Final approximations: first and second
bool init=false;

current.f=null; // Save to preserve the original
current.s=end;

if(VERBOSE_DEBUG)
    cout<<"Before_while"
        <<endl
        <<"NEXT:_logsearch(while)"<<PPrintPoint3("current.f",current.f,false)
        <<endl
        <<"NEXT:_logsearch(while)"<<PPrintPoint3("current.s",current.s,false)
        <<endl
        <<"NEXT:_logsearch(while)"<<PPrintPoint3("pm",pm,false)

```

```

        <<endl
        <<" while"<<endl;

// If we find an intersection, then we're done
while (!onZinz(pm))
{
    // null.t -> (current.f.t+current.s.t)/2
    pm=solve((current.f.t+current.s.t)/2,null);

    // Was a good interval found? -> Update the respective one
    if (firstGoodZ (pm)) current.f=pm;
    if (secondGoodZ(pm)) current.s=pm;

    if(VERBOSE_DEBUG)
        cout<<"NEXT: _logsearch (while)"<<PPrintPoint3("pm",pm, false)
            <<endl
            <<"NEXT: _logsearch (while)"<<PPrintPoint3("current.f",current.f, false)
            <<endl
            <<"NEXT: _logsearch (while)"<<PPrintPoint3("current.s",current.s, false)
            <<endl;
} // While

// Intersects Z
if(VERBOSE_DEBUG)
    cout<<"Intersects_Z:"<<PPrintPoint3("current.f",current.f, false)<<endl
        <<"Intersects_Z:"<<PPrintPoint3("current.s",current.s, false)<<endl;

pm2=pm; // This will be the absolute middle

//-----

// Trying to put closer the first interval to the middle interval

init=true;

// If we find an intersection, then we're done
while (init || (!onZinz(pm) && firstBetterZ (pm2,pm)))
{
    init=false;
    // null.t -> (current.f.t+pm2.t)/2
    pm=solve((current.f.t+pm2.t)/2,null);

    // If the new one is better, update. If not, leave it.
    // Still good AND better then the original
    if (firstGoodZ (pm) && firstBetterZ (current.f,pm)) current.f=pm;

    if(VERBOSE_DEBUG)
        cout<<"NEXT: _logsearch (while2)"<<PPrintPoint3("pm",pm, false)
            <<endl
            <<"NEXT: _logsearch (while2)"<<PPrintPoint3("current.f",current.f, false)
            <<endl;
} // While

// Intersects Z
if(VERBOSE_DEBUG)
    cout<<"Intersects_Z_2(while)"<<PPrintPoint3("current.f",current.f, false)
        <<endl

```

```

        <<"Intersects_Z_2(while)"<<PPrintPoint3("current.s",current.s,false)
        <<endl;

//-----

// Trying to put closer the first interval to the middle interval

init=true;

// If we find an intersection, then we're done
while (init || (!onZinz(pm) && secondBetterZ(pm2,pm)))
{
    init=false;
    // null.t -> (current.s.t+pm2.t)/2
    pm=solve((current.s.t+pm2.t)/2,null);

    // If the new one is better, update. If not, leave it.
    // Still good AND better then the original
    if (secondGoodZ(pm) && secondBetterZ(current.s,pm)) current.s=pm;

    if(VERBOSE_DEBUG)
        cout<<"NEXT:_logsearch(while3)"<<PPrintPoint3("pm",pm,false)
        <<endl
        <<"NEXT:_logsearch(while3)"<<PPrintPoint3("current.s",current.s,false)
        << endl;
} // While

// Intersects Z
if(VERBOSE_DEBUG)
    cout<<"Intersects_Z_3(while)"<<PPrintPoint3("current.f",current.f,false)
    <<endl
    <<"Intersects_Z_3(while)"<<PPrintPoint3("current.s",current.s,false)
    <<endl;

return current;
} // LogSearch

//-----

// Solve at a given t from initial value p
point3 solve(interval tend,point3 p){
VNODE* Solver=VE.Solver;
point3 p2=p;

iVector y(3);
y=point3ToiVector(p);

interval t=p.t;

Solver->setFirstEntry();
Solver->setHmin(0);
Solver->setOneStep(on);
while(t!=tend)
{
    Solver->integrate(t,y,tend);
    if(Solver->successful()&& Solver->getGlobalExcess()<=VE.MaxGlobalExcess)
    {

```

```

        p2=iVectorTopoint3(y,t);
        if(VERBOSE_DEBUG) print(p2,Solver->getGlobalExcess());
    }
    else
    {
        cout << "FATÁL_ERROR(solve)!" <<endl;
        exit(1);
    }
} // While

return p2;
}

```

```

void header(){

    cout << "midpoint(t)" <<"\t"
        << "midpoint(x)" <<"\t"
        << "midpoint(y)" <<"\t"
        << "midpoint(z)" <<"\t"
        << "rad(t)" <<"\t"
        << "rad(x)" <<"\t"
        << "rad(y)" <<"\t"
        << "rad(z)" <<"\t"
        << "Solver->getGlobalExcess()"
        << endl;
} // header

void print(point3 p, double GlobalExcess){

    if (PRINT2)
    {
        cout << "t (" << LEnd(p.t) << ", " << UEnd(p.t) << ") "
            << "\tx (" << LEnd(p.x) << ", " << UEnd(p.x) << ") "
            << "\ty (" << LEnd(p.y) << ", " << UEnd(p.y) << ") "
            << "\tz (" << LEnd(p.z) << ", " << UEnd(p.z) << ") "
            << "\t" << GlobalExcess
            << endl;
    }
    else{
        cout << midpoint(p.t) <<"\t"
            << midpoint(p.x) <<"\t"
            << midpoint(p.y) <<"\t"
            << midpoint(p.z) <<"\t"
            << rad(p.t) <<"\t"
            << rad(p.x) <<"\t"
            << rad(p.y) <<"\t"
            << rad(p.z) <<"\t"
            << GlobalExcess
            << endl;
    }
} // print

iVector point3ToiVector(point3 p){

iVector y(3);

```

```

y[0]=p.x;
y[1]=p.y;
y[2]=p.z;
return y;
} // point3ToiVector

point3 iVectorTopoint3(iVector y,interval t){

point3 p;
p.t=t;
p.x=y[0];
p.y=y[1];
p.z=y[2];
return p;
} // iVectorTopoint3

double LEnd(interval x){ // Gives the lower end of an interval
return midpoint(x)-rad(x);
}

double UEnd(interval x){ // Gives the upper end of an interval
return midpoint(x)+rad(x);
}

string PPrintPoint3(string name, point3 p,bool with_t){
stringstream s;

s.setf(ios::fixed,ios::floatfield);
s.precision(20);

if (with_t) s << name <<"t(" << LEnd(p.t) <<"," << UEnd(p.t) <<")\t";

s << name <<"x(" << LEnd(p.x) <<"," << UEnd(p.x) <<")\t"
<< name <<"y(" << LEnd(p.y) <<"," << UEnd(p.y) <<")\t"
<< name <<"z(" << LEnd(p.z) <<"," << UEnd(p.z) <<");

return s.str();
}

string PPrintPoint3WOZ(string name, point3 p,bool with_t){
stringstream s;

s.setf(ios::fixed,ios::floatfield);
s.precision(20);

if (with_t) s << name <<"t(" << LEnd(p.t) <<"," << UEnd(p.t) <<")\t";

s << name <<"x(" << LEnd(p.x) <<"," << UEnd(p.x) <<")\t"
<< name <<"y(" << LEnd(p.y) <<"," << UEnd(p.y) <<");

return s.str();
}

string PPrintGnuPlot(point3 p,bool with_t){
stringstream s;

s.setf(ios::fixed,ios::floatfield);

```

```

s.precision(20);

s << midpoint(p.x) <<"_ "<< midpoint(p.y) <<"_ "
<< LEnd(p.x) <<"_ "<< UEnd(p.x) <<"_ "
<< LEnd(p.y) <<"_ "<< UEnd(p.y) <<"_ #"
<< LEnd(p.z) <<"_ "<< UEnd(p.z) ;

if (with_t)
s <<"_ "<< LEnd(p.t) <<" , "<< UEnd(p.t) <<" )";

return s.str();
}

//-----

static const struct T_R{
interval m,br,bl,bt,bb;
T_R(){
interval m_szaml=0.0036;
interval m_nev =0.0496;
m= m_szaml/m_nev;
br=1.686;
bl=1.7356;
bt=-0.061 + m*1.686;
bb=-0.063 + m*1.686;
}
} R;

static const struct T_L{
interval mr,mt,trx,brx,mb,br,bl,bt,bb;
T_L(){
interval mr_szaml= 0.00128;
interval mr_nev =-0.017;
mr= mr_szaml/mr_nev;
interval mt_szaml=0.00182;
interval mt_nev =0.022;
mt= mt_szaml/mt_nev;
interval mb_szaml=0.00254;
interval mb_nev =0.039;
mb= mb_szaml/mb_nev;

trx=1.839; // Top right x
brx=1.822; // Bottom right x
br=-0.07218 + mr*1.839;
bl=1.861;
bt=-0.07218 + mt*1.839;
bb=-0.07346 + mb*1.822;
}
} L;

bool lOFrinRx(point3 p){
return (UEnd(p.x + R.br) < 0);
}

bool rOFlinRx(point3 p){
return (LEnd(p.x + R.bl) > 0);
}

```

```

bool rOFrinRx(point3 p){ // The whole interval is to right of R
return (LEnd(p.x + R.br) > 0);
}

bool lOFlinRx(point3 p){ // The whole interval is to left of R
return (UEnd(p.x + R.bl) < 0);
}

bool inRx(point3 p){
return lOFrinRx(p) && rOFlinRx(p);
}

bool NOTinRx(point3 p){ // The whole interval is to the right
return rOFrinRx(p) || lOFlinRx(p); // OR to the left of R
}

// top bottom upper lower // 0< equation -> Lower then line
// lowerside OF top // 0> equation -> Upper then line
bool lOFtinRy(point3 p){
return (UEnd( p.y - R.m*p.x - R.bt) < 0);
}

// top bottom upper lower
// upperside OF bottom
bool uOFbinRy(point3 p){
return (LEnd(p.y - R.m*p.x - R.bb) > 0);
}

// top bottom upper lower
// upperside OF top
bool uOFtinRy(point3 p){ // The whole interval is above R
return (LEnd(p.y - R.m*p.x - R.bt) > 0);
}

// top bottom upper lower
// lowerside OF bottom
bool lOFbinRy(point3 p){ // The whole interval is under R
return (UEnd(p.y - R.m*p.x - R.bb) < 0);
}

bool inRy(point3 p){
return lOFtinRy(p) && uOFbinRy(p);
}

bool NOTinRy(point3 p){ // The whole interval is above OR under R
return uOFtinRy(p) || lOFbinRy(p);
}

bool NOTinR(point3 p){
// If the whole interval is NOT in R wrt x [y], then y [x] doesn't matter
// If the interval intersects R wrt x then it could intersect the Taboo set
too!
// This is checked elsewhere
return NOTinRx(p) || NOTinRy(p);
}

```

```

bool NOTinRT(point3 p){ // NOT in the Taboo set of R
// If the whole interval is in R wrt y, it's not in the Taboo set OR
// If the whole interval is NOT in R wrt x there's no need to check further
return inRy(p) || NOTinRx(p);
}

bool lOFrinLx(point3 p){
return (UEnd(p.y - L.mr*p.x - L.br) < 0);
}

bool rOFlinLx(point3 p){
return (LEnd(p.x + L.bl) > 0);
}

bool rOFrinLx(point3 p){ // The whole interval is to right of L
return (LEnd(p.y - L.mr*p.x - L.br) > 0);
}

bool lOFlinLx(point3 p){ // The whole interval is to left of L
return (UEnd(p.x + L.bl) < 0);
}

// Farther then 10-8 from line C. This is for SLOPING line C
bool FAROFrinLx(point3 p){
interval lf = p.y - L.mr*p.x - L.br + interval(-1,1)*1.0E-8;
return (UEnd(lf) < 0) || (LEnd(lf) > 0);
}

bool lOFtrinLx(point3 p){ // To the left of the right end of the shorter
line
return (UEnd(p.x + L.trx) < 0); // by x in L
}

bool rOFtrinLx(point3 p){ // To the right of the right end of the shorter
line
return (LEnd(p.x + L.trx) > 0); // by x in L
}

bool rOFbrinLx(point3 p){ // To the right of the right end of the longer
line
return (LEnd(p.x + L.brx) > 0); // by x in L
}

// Not in L Vertically (=the longer line by x): No need to check the shorter
bool NOTinLvx(point3 p){
return lOFlinLx(p) || rOFbrinLx(p);
}

bool inLx(point3 p){
return lOFrinLx(p) && rOFlinLx(p);
}

bool NOTinLx(point3 p){
return rOFrinLx(p) || lOFlinLx(p);
}

// top bottom upper lower

```

```

// lowerside OF top
bool lOFtinLy(point3 p){
return (UEnd(p.y - L.mt*p.x - L.bt) < 0);
}

// top bottom upper lower
// upperside of bottom
bool uOFbinLy(point3 p){
return (LEnd(p.y - L.mb*p.x - L.bb) > 0);
}

// top bottom upper lower
// upperside OF top
bool uOFtinLy(point3 p){ // The whole interval is above L
return (LEnd(p.y - L.mt*p.x - L.bt) > 0);
}

// top bottom upper lower
// lowerside OF bottom
bool lOFbinLy(point3 p){ // The whole interval is under L
return (UEnd(p.y - L.mb*p.x - L.bb) < 0);
}

bool inLy(point3 p){
return lOFtinLy(p) && uOFbinLy(p);
}

bool NOTinLy(point3 p){
return uOFtinLy(p) || lOFbinLy(p);
}

bool NOTinL(point3 p){
return NOTinLx(p) || NOTinLy(p);
}

bool NOTinLT(point3 p){ // NOT in the Taboo set of L
// The whole interval is in the VERTICAL cover by the LONGER line AND
// The whole interval is in L wrt y
// OR The whole interval is to the right of the shorter line wrt x AND
// Intersects the VERTICAL COVER AND above the lower line
// (only the lower counts) OR Disjoint from the VERTICAL COVER
return (!lOFlinLx(p) && !rOFbrinLx(p) && inLy(p)) ||
(rOFtrinLx(p) && !rOFbrinLx(p) && uOFbinLy(p)) ||
lOFlinLx(p) || rOFbrinLx(p) ;
}

bool uZ(point3 p){ // Under Z AND in Tol
return (LEnd(p.z) > UEnd(VE.z - VE.zTol)) &&
(UEnd(p.z) < LEnd(VE.z)) ;
}

bool aZ(point3 p){ // Above Z AND in Tol
return (LEnd(p.z) > UEnd(VE.z)) &&
(UEnd(p.z) < LEnd(VE.z + VE.zTol)) ;
}

bool oZ(point3 p){ // ON Z AND in Tol

```

```

return (LEnd(p.z) > UEnd(VE.z - VE.zTol)) &&
        (UEnd(p.z) < LEnd(VE.z + VE.zTol)) ;
}

bool aOFZinz(point3 p){ // Above Z
return (LEnd(p.z) > UEnd(VE.z));
}

bool uOFZinz(point3 p){ // Under Z
return (UEnd(p.z) < LEnd(VE.z));
}

bool onZinz(point3 p){ // ON Z (Intersects Z)
return (!aOFZinz(p) && !uOFZinz(p));
}

bool IsSame(point3 p1, point3 p2){
return
// If the two approximations are not at the same "side" of the quadrangles
// check the stricter rules for both!
(lOFrinRx(p1) == lOFrinRx(p2) || ((!NOTinRx(p1) || !NOTinRx(p2)) &&
    inRy(p1) && inRy(p2))) &&
(rOFlinRx(p1) == rOFlinRx(p2) || ((!NOTinRx(p1) || !NOTinRx(p2)) &&
    inRy(p1) && inRy(p2))) &&
(rOFrinRx(p1) == rOFrinRx(p2) || ((!NOTinRx(p1) || !NOTinRx(p2)) &&
    inRy(p1) && inRy(p2))) &&
(lOFlinRx(p1) == lOFlinRx(p2) || ((!NOTinRx(p1) || !NOTinRx(p2)) &&
    inRy(p1) && inRy(p2))) &&
(lOFtrinLx(p1) == lOFtrinLx(p2) || ((!NOTinLvx(p1) || !NOTinLvx(p2)) &&
    inLy(p1) && inLy(p2))) &&
(rOFlinLx(p1) == rOFlinLx(p2) || ((!NOTinLvx(p1) || !NOTinLvx(p2)) &&
    inLy(p1) && inLy(p2))) &&
(rOFbrinLx(p1) == rOFbrinLx(p2) || ((!NOTinLvx(p1) || !NOTinLvx(p2)) &&
    inLy(p1) && inLy(p2))) &&
(lOFlinLx(p1) == lOFlinLx(p2) || ((!NOTinLvx(p1) || !NOTinLvx(p2)) &&
    inLy(p1) && inLy(p2))) &&
// These are the wertical checks
((NOTinRx(p1) && NOTinRx(p2)) || lOFtinRy(p1) == lOFtinRy(p2)) &&
((NOTinRx(p1) && NOTinRx(p2)) || uOFbinRy(p1) == uOFbinRy(p2)) &&
((NOTinRx(p1) && NOTinRx(p2)) || uOFtinRy(p1) == uOFtinRy(p2)) &&
((NOTinRx(p1) && NOTinRx(p2)) || lOFbinRy(p1) == lOFbinRy(p2)) &&
((NOTinLvx(p1) && NOTinLvx(p2)) || lOFtinLy(p1) == lOFtinLy(p2)) &&
((NOTinLvx(p1) && NOTinLvx(p2)) || uOFbinLy(p1) == uOFbinLy(p2)) &&
((NOTinLvx(p1) && NOTinLvx(p2)) || uOFtinLy(p1) == uOFtinLy(p2)) &&
((NOTinLvx(p1) && NOTinLvx(p2)) || lOFbinLy(p1) == lOFbinLy(p2)) &&
// The sloping right-hand side of L needs a delicate care!
// In the Vertical cover
((NOTinLvx(p1) && NOTinLvx(p2)) || lOFrinLx(p1) == lOFrinLx(p2) ||
    ((!NOTinLvx(p1) || !NOTinLvx(p2)) && inLy(p1) && inLy(p2))) &&
((NOTinLvx(p1) && NOTinLvx(p2)) || rOFrinLx(p1) == rOFrinLx(p2) ||
    ((!NOTinLvx(p1) || !NOTinLvx(p2)) && inLy(p1) && inLy(p2))) &&
// Check for Z too: p1 should be under/above Z AND p2 should be above/under Z
((uZ(p1) && aZ(p2)) || (uZ(p2) && aZ(p1))) ;
}

//-----

```

```

int main(){
// Just for precision
cout.setf(ios::fixed,ios::floatfield);
cout.precision(20);
cerr.setf(ios::fixed,ios::floatfield);
cerr.precision(20);
cerr << "zTol:" << VE.zTol<< endl;

interval tnull= 0.0;
interval eps = interval(-1,1)*1.0E-8;

//A1-2: (-1.686,-0.063),(-1.686,-0.061)
//B1-2: (-1.7356,-0.0666),(-1.7356,-0.0646)
//C1-2: (-1.839,-0.07218),(-1.822,-0.07346)
//D1-2: (-1.861,-0.076),(-1.861,-0.074)

point3 test;
test.t = tnull;
test.z = 0.3 + eps;

//L: Cut to 10 pieces by x by 0.0039

//1.
//~ test.x = interval(-1.861,-1.8571);
//~ test.y = interval(-0.076,-0.07218);

//2.
//~ test.x = interval(-1.8571,-1.8532);
//~ test.y = interval(-0.076,-0.07218);

//3.
//~ test.x = interval(-1.8532,-1.8493);
//~ test.y = interval(-0.076,-0.07218);

//4.
//~ test.x = interval(-1.8493,-1.8454);
//~ test.y = interval(-0.076,-0.07218);

//5.
//~ test.x = interval(-1.8454,-1.8415);
//~ test.y = interval(-0.076,-0.07218);

//6.
//~ test.x = interval(-1.8415,-1.8376);
//~ test.y = interval(-0.076,-0.07218);

//7.
//~ test.x = interval(-1.8376,-1.8337);
//~ test.y = interval(-0.076,-0.07218);

//8.
//~ test.x = interval(-1.8337,-1.8298);
//~ test.y = interval(-0.076,-0.07218);

//9.
//~ test.x = interval(-1.8298,-1.8259);
//~ test.y = interval(-0.076,-0.07218);

```

```

//10.
//~ test.x = interval(-1.8259,-1.822);
//~ test.y = interval(-0.076,-0.07218);

//-----

// L
//~ test.x = interval(-1.861,-1.822);
//~ test.y = interval(-0.076,-0.07218);

// R
//~ test.x = interval(-1.7356,-1.686);
//~ test.y = interval(-0.0666,-0.061);

// A line
//~ test.x = interval(-1.686,-1.686);
//~ test.y = interval(-0.063,-0.061);

// B line
//~ test.x = interval(-1.7356,-1.7356);
//~ test.y = interval(-0.0666,-0.0646);

// C line
//~ test.x = interval(-1.839,-1.822);
//~ test.y = interval(-0.07346,-0.07218);

// D line
//~ test.x = interval(-1.861,-1.861);
//~ test.y = interval(-0.076,-0.074);

vector<point3> ret;

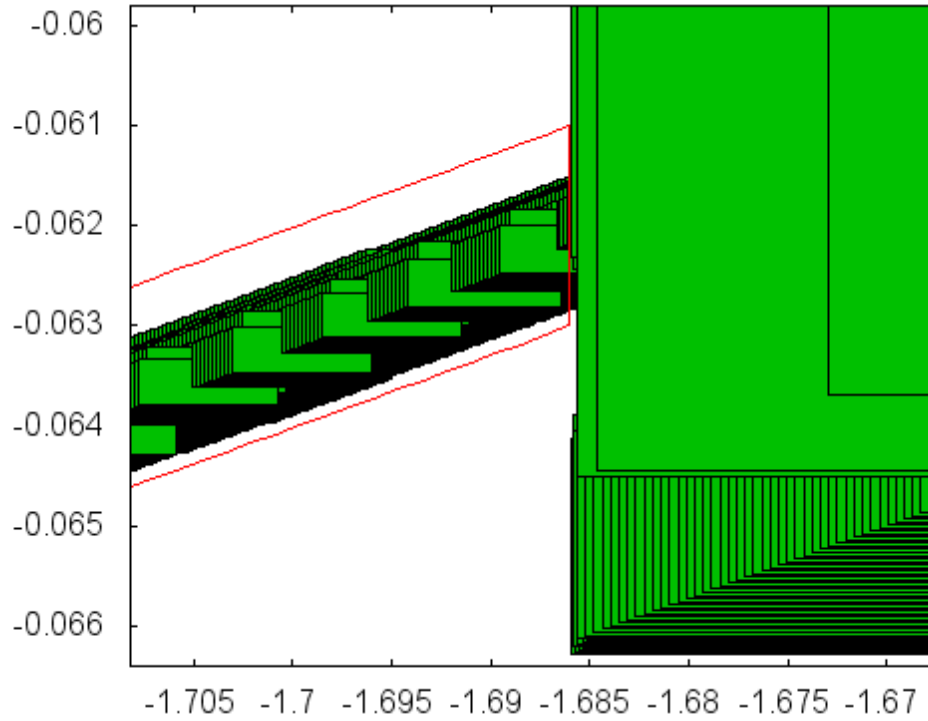
ret = BB(test,1);
cout << "Final_num._of_pieces:"<< ret.size() << endl;
cerr << "Final_num._of_pieces:"<< ret.size() << endl;

return 0;
}

```

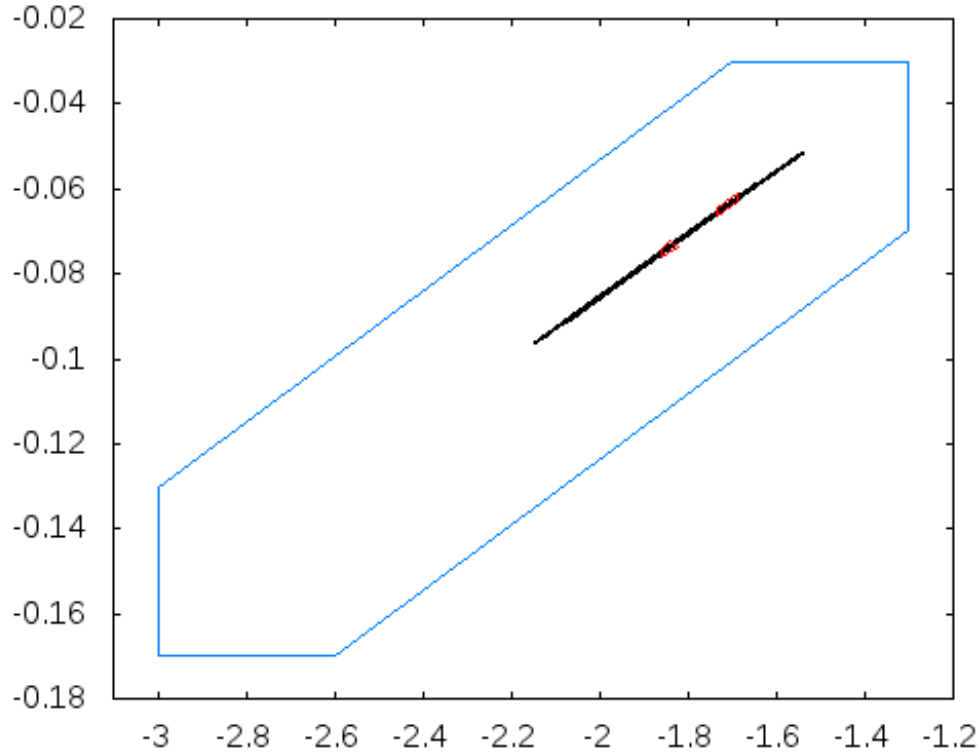
4. Számítógépes ábrák

A most következő ábrák a 2.3 alfejezet legvégén megfogalmazott $(\mathcal{C})$ feltétel VNODE általi ellenőrzést demonstrálják. Az első két ábra a hozzájuk fűzött magyarázatok miatt folytatólagos sorszámozású.



10. Ábra: Teljesíthető-e a (2) feltétel? — Kép-téglatestek vetületei az R négyszög a oldalához közel

A 10. Ábra — csak úgy mint a 2.3 alfejezet 8. Ábrája — egyszerre kapcsolatos a (12) és a (2) tulajdonság teljesülésével. Olyan $\text{proj}K_k$ téglalapcsaládot mutat, amely az R négyszög a oldalához közel helyezkedik el, s amelynek egyetlen eleme sem metsz bele a 2. Ábrán látható E tabu halmazba. A mögöttes $\{T_k\}$ tárgy-téglalapcsalád egyébként az L négyszög c oldalához közeli "csőrének" egy részét fedi le.

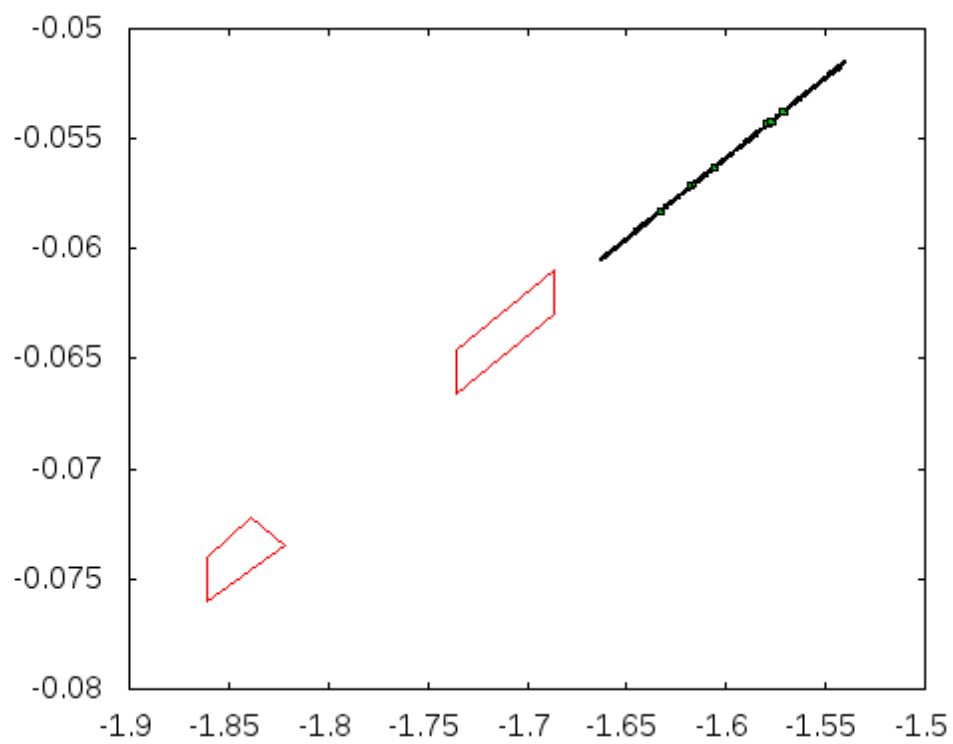


11. Ábra: Az utolsó ellenőrzés: a $\varphi(L \cup R)$ halmaz elhelyezkedése a H hatszögben

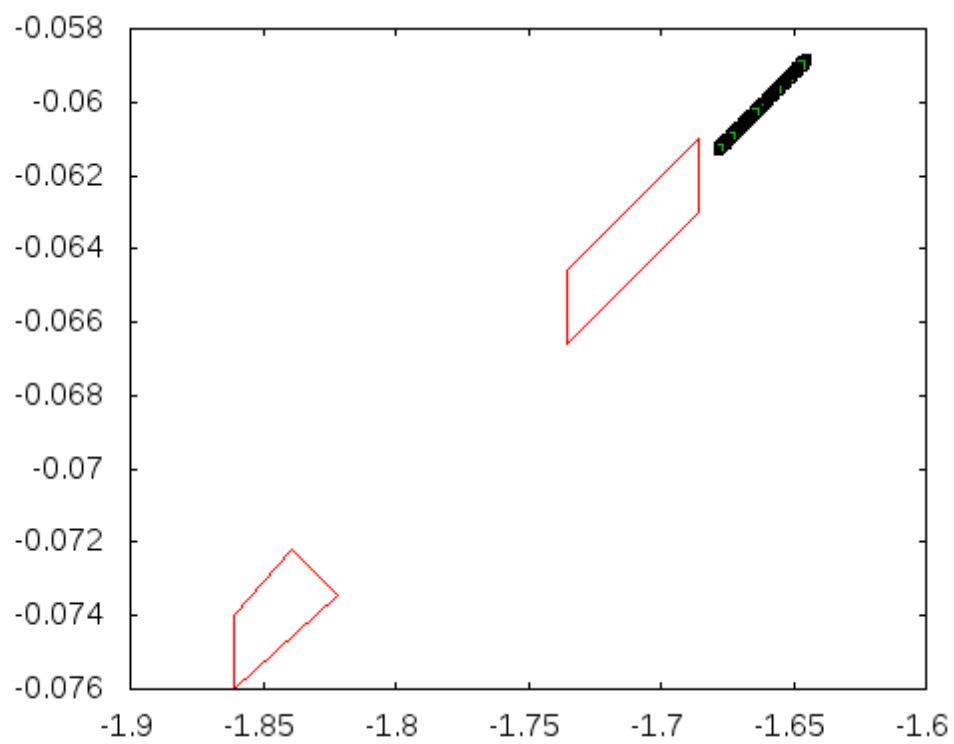
A 11. Ábra a $(\mathcal{C})$ tulajdonság teljesülését ellenőrző VNODE számítások összességére utal. Ennek az ábrának a jellege azért különbözik a korábbiaktól, mert az utolsó, a tényleges bizonyítást jelentő futtatás előtt az átmetszési időintervallumot lehetőség szerint minél kisebbre kényszerítendő azt írtam elő, hogy $z = 0.3$ sík alatt elhelyezkedő *első* téglatest a $z = 0.2999$ sík fölé, a $z = 0.3$ sík felett elhelyezkedő *második* téglatest pedig a $z = 0.3001$ sík alá kerüljön. Ez volt az első, nem-kísérletező jellegű olyan futtatás is, amelyben a $\text{proj}K_k$ halmazok helyett ténylegesen a

$$\text{proj}K_k + [0, \tau_k] \cdot \text{proj}f(K_k) + ([-65, 15] \times [-4, 2])\tau_k^2$$

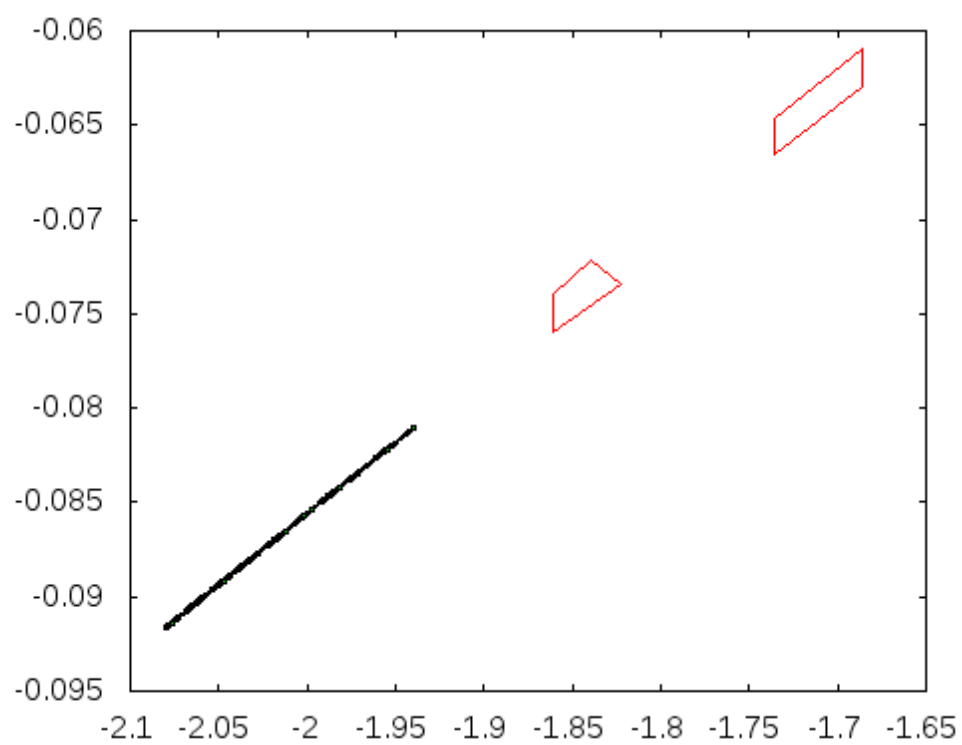
halmazok szerepeltek.



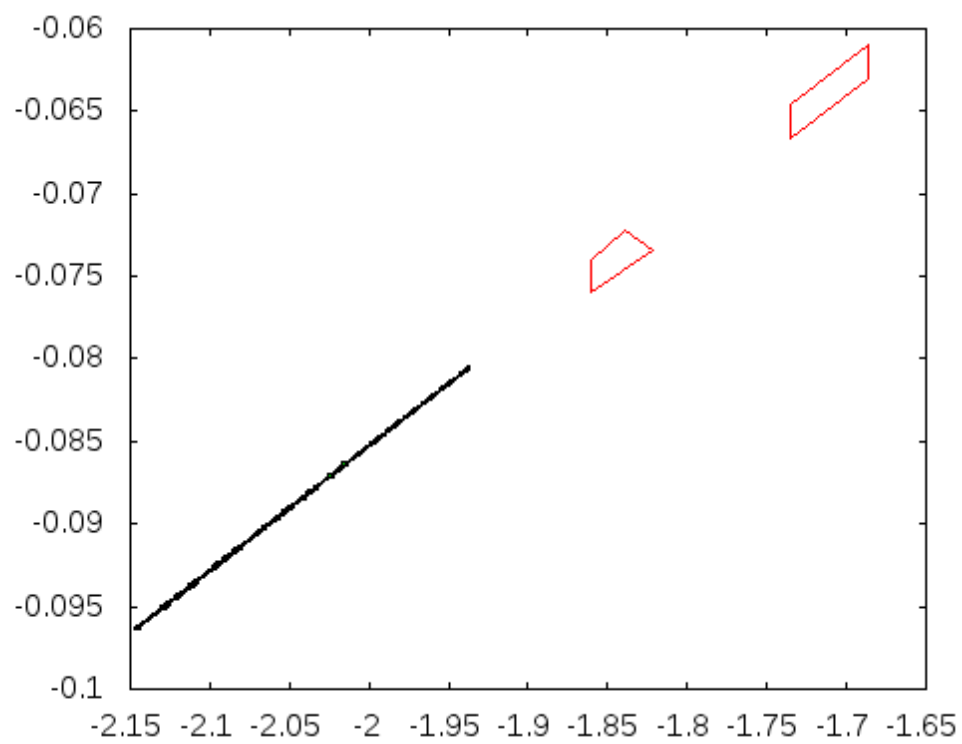
A $\varphi(a) \subset O_R$ feltétel ellenőrzése
 Futási idő: < 20 másodperc
 Intervallum-felosztások száma: < 150
 Maximális veremmélység: < 9



A $\varphi(c) \subset O_R$ feltétel ellenőrzése
 Futási idő: < 8 perc
 Intervallum-felosztások száma: < 10000
 Maximális veremmélység: < 19



A $\varphi(b) \subset O_L$ feltétel ellenőrzése
 Futási idő: < 15 másodperc
 Intervallum-felosztások száma: < 200
 Maximális veremmélység: < 9

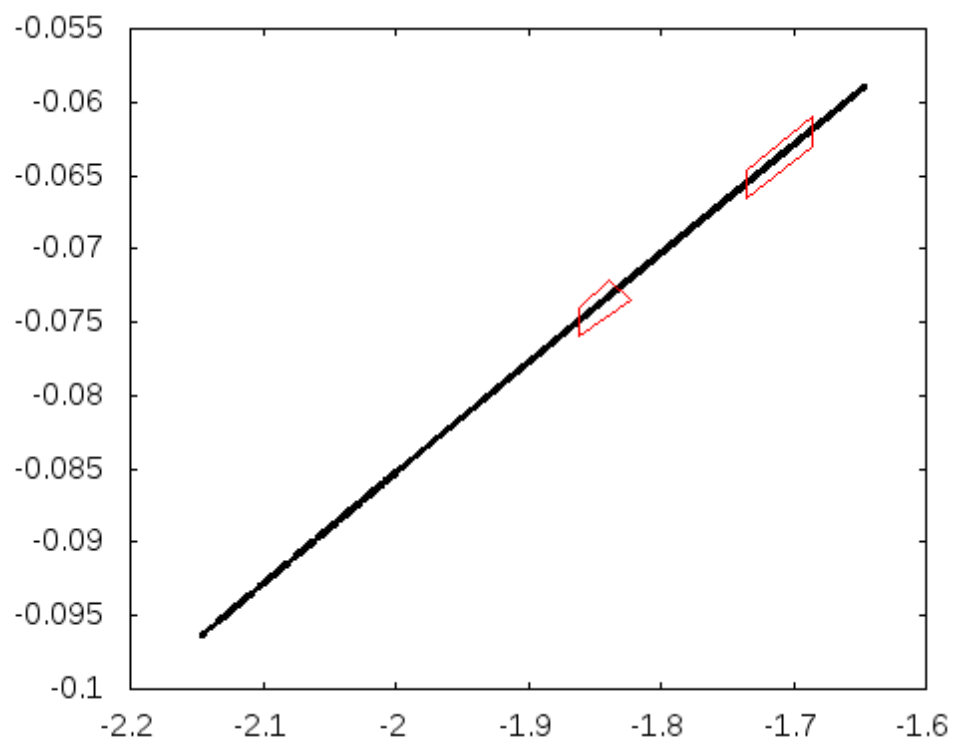


A $\varphi(d) \subset O_L$ feltétel ellenőrzése

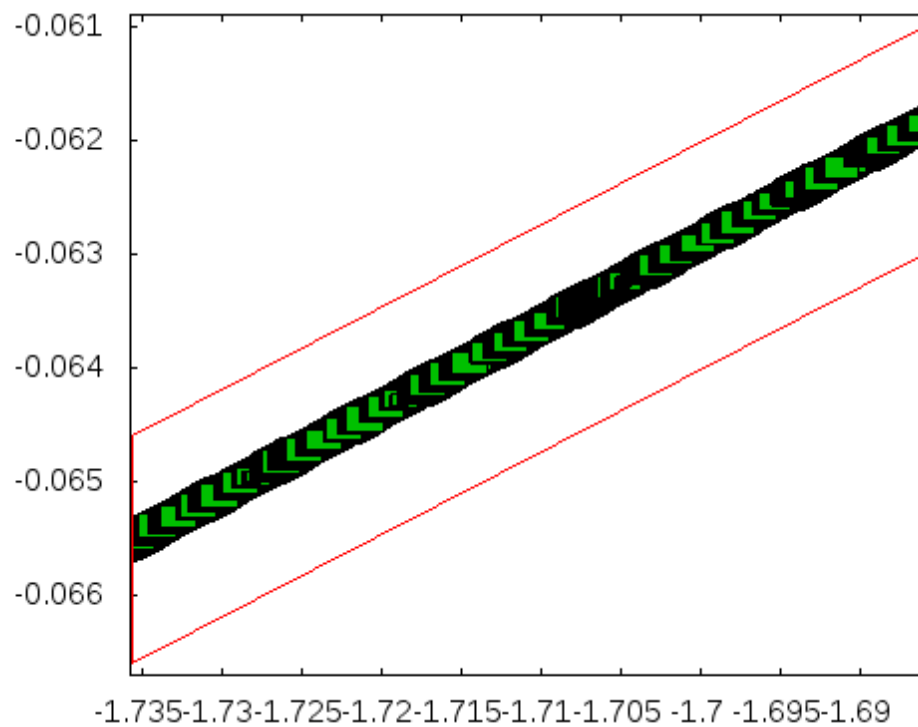
Futási idő: < 1 perc

Intervallum-felosztások száma: < 600

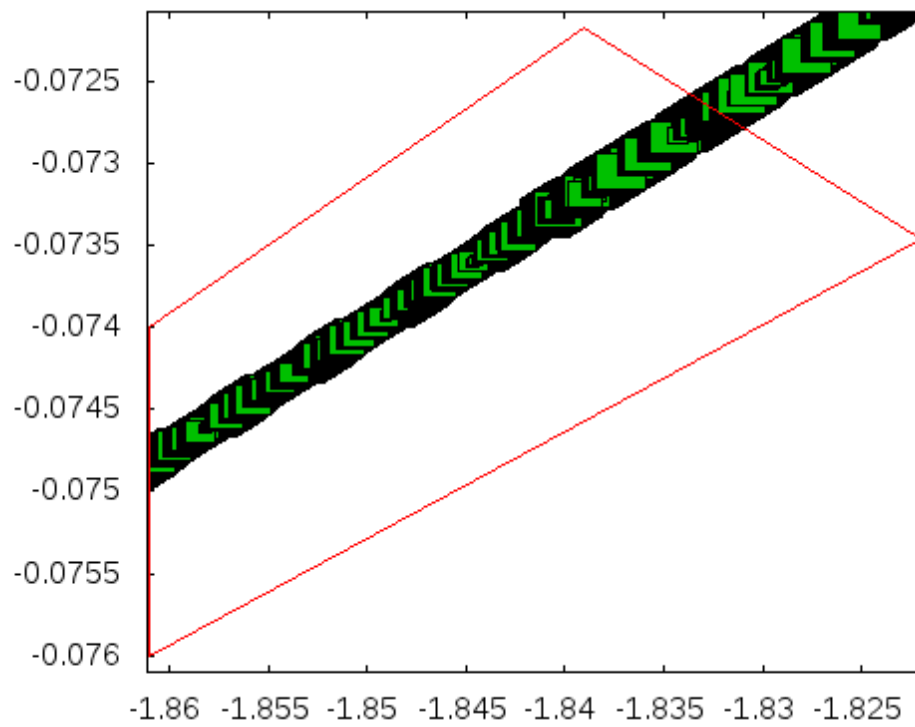
Maximális veremmélység: < 11



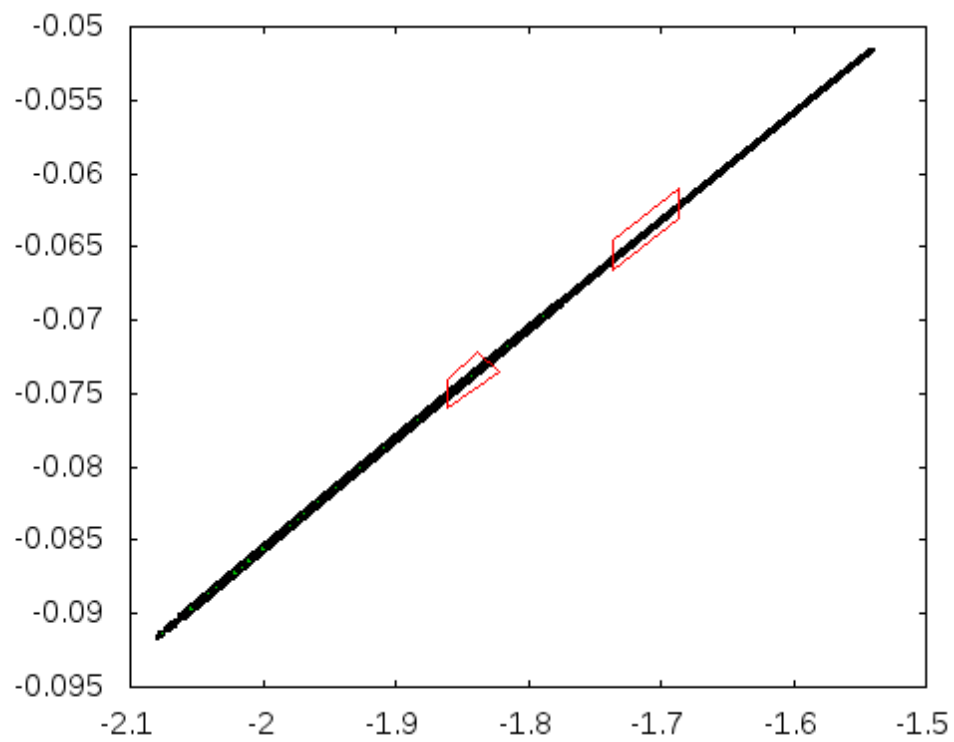
A $\varphi(L) \cap E = \emptyset$ feltétel ellenőrzése
 Futási idő: < 12000 perc
 Intervallum-felosztások száma: < 8700000
 Maximális veremmélység: < 23



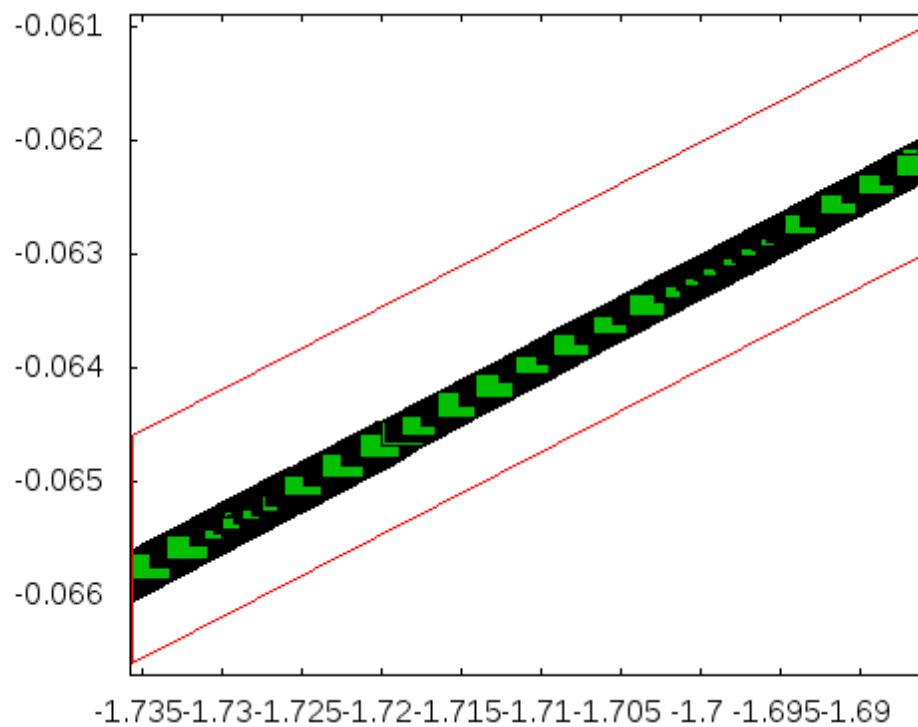
A $\varphi(L) \cap E = \emptyset$ feltétel ellenőrzése
 Futási idő: < 12000 perc
 Intervallum-felosztások száma: < 8700000
 Maximális veremmélység: < 23



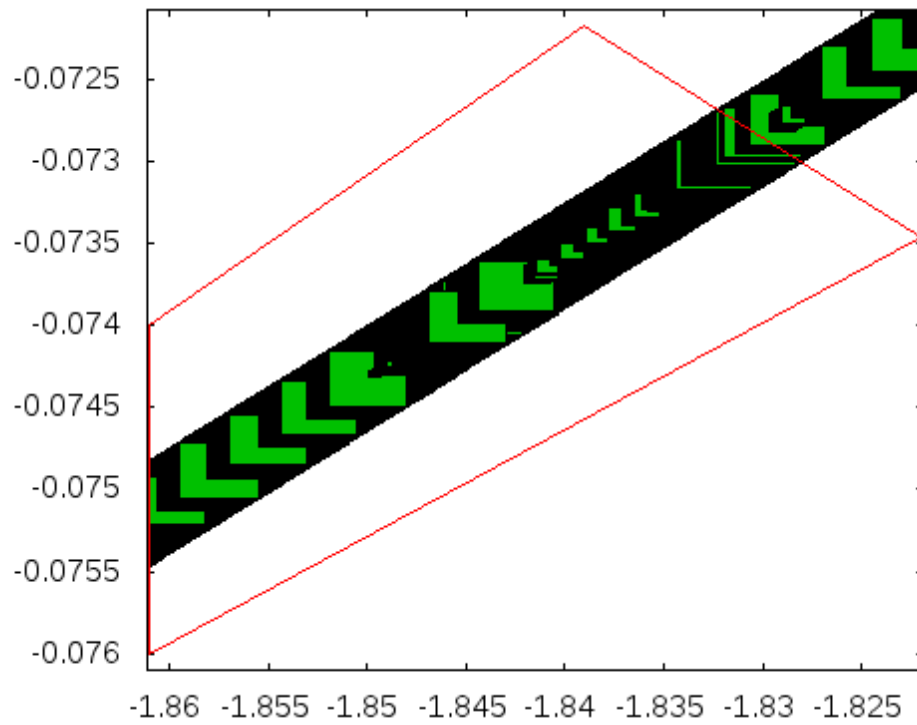
A $\varphi(L) \cap E = \emptyset$ feltétel ellenőrzése
 Futási idő: < 12000 perc
 Intervallum-felosztások száma: < 8700000
 Maximális veremmélység: < 23



A $\varphi(R) \cap E = \emptyset$ feltétel ellenőrzése
 Futási idő: < 875 perc
 Intervallum-felosztások száma: < 450000
 Maximális veremmélység: < 23



A $\varphi(R) \cap E = \emptyset$ feltétel ellenőrzése
 Futási idő: < 875 perc
 Intervallum-felosztások száma: < 450000
 Maximális veremmélység: < 23



A $\varphi(R) \cap E = \emptyset$ feltétel ellenőrzése
 Futási idő: < 875 perc
 Intervallum-felosztások száma: < 450000
 Maximális veremmélység: < 23