Függvényegyenletek Regularitási Tulajdonságainak Vizsgálata Számítógéppel

Czirbusz Sándor

2015. december 4.

Függvényegyenletek Regularitási Tulajdonságainak
Vizsgálata Számítógéppel

Témavezető: Dr. Járai Antal

Ezúton szeretnék köszönetet mondani mindazoknak, akik segítettek dolgozatom létrejöttében. Először Dr. Járai Antal egykori tanáromnak és témavezetőmnek, hogy éveken keresztül segítette és támogatta munkámat. Ezen felül minden kollégámnak is köszönöm a türelmet.
Természetesen köszönet illeti a feleségemet és családomat is az éveken át tartókitartásáért.
Legvégül köszönöm a közben nyugdíjba vonult titkárnőknek is bizalmat.

TARTALOMJEGYZÉK

BEVEZETÉS

A függvényegyenletekkel mind a tiszta matematikában, mind alkalmazásaiban – közgazdaság, fizika – gyakran találkozhatunk. Megoldásuk a matematikai igen régi és nehéz kihívásai közé tartozik. A terület igazán önálló diszciplínává csak a XX. században vált, az első nagy összegzés Aczél János [5]-ben művében történt meg. Olyan átfogó elmélettel, mind például a közönséges differenciálegyenletek elmélete, mind a mai napig nem rendelkezik. Emellett elvétve találunk csak olyan műveket, amelyek egy bevezető egyetemi kurzus alapját képezhetnék, kivételek talán a Efthimiou [22], vagy a Small [38] művek. Ennek legfontosabb oka a terület összetettsége, az alkalmazott módszerek sok esetben jelentős matematikai apparátust igényelnek, ötvözik az analízis elemi és haladó módszereit, a mértékelméletet, funkcionálanalízis, valamint az absztrakt algebra sok eszközét. Tágabb értelemben függvényegyenletnek tekinthetjük a differenciál- és integrálegyenleteket, rekurzív sorozatokat is, a diszciplína szokásos konvenciója azonban ezeket az egyenleteket kizárja a függvényegyenletek köréből. Az irodalomban találhatunk definiálási és kategorizálási kísérleteket (Aczél [5], Kuczma [31]), ezek azonban többnyire nem alkalmasak arra, hogy gyakorlatban hasznosíthassuk őket, lásd Daróczy [21] cikkét. A nehézség egyik oka magában a függvény fogalmában található; már az alsóéves egyetemisták számára is természetes a függvény halmazelméleti definiálása, azaz egy $f : A \to B$ függvény olyan részhalmaza az $A \times B$ halmaznak, melyre $(x, y_1) \in f$ és $(x, y_2) \in f$ esetén $y_1 = y_2$. A függvényegyenletek esetén azonban gyakran úgy tekintünk a függvényekre, mint formális szabályokra (Így találkozik a függvény fogalmával például a középiskolás is). Ez a függvényfogalom a számítógépes felfogáshoz is közelebb áll. A függvényegyenletekkel kapcsolatban azonban nem túl sok számítógéppel támogatott megoldás, alkalmazás létezik.

A dolgozat első részében megvizsgáljuk a jelenlegi számítógépes alkalmazásokat, lehetőségeket, azokat példákkal is illusztrálva. Ismertetünk egy egyszerű egyenletosztályra alkalmazható, általunk kifejlesztett algoritmust is.
A függvényegyenletek kezelését, megoldását bonyolítja, hogy sok esetben az általános

megoldások illetve a valamilyen analitikus feltételnek eleget tevő „reguláris” megoldások (mérhető, folytonos, differenciálható, stb.) között óriási különbség van. Erre az additív Cauchy egyenlet a legegyszerűbb példa. A dolgozat második részében a regularitás elméleti hátterével foglalkozunk. Ebből a szempontból részletesen vizsgáljuk az

$$f(x) = h\Big(x, y, f\big(g_1(x,y)\big), \ldots, f\big(g_n(x,y)\big)\Big)$$

típusú egyenleteket, ahol $h, g_1, \ldots, g_n$ ismert függvények, f az ismeretlen függvény, x, y pedig valamilyen (nem feltétlenül azonos) euklideszi térből való, az előző függvények értékkészlete szintén valamilyen euklideszi tér nyílt részhalmaza. Ezt a témakört részletesen tárgyalja és majdnem teljes elméleti megoldást nyújt Járai [29].

A dolgozat harmadik fejezete a szimbolikus intervallum aritmetikával foglalkozik, mely kifejlesztésével az egyik célunk az, hogy a függvényegyenletek algebrai átalakítása közben kiszámoljuk a módosított függvények értelmezési tartományát illetve értékkészletét. Ezen kívül a továbbiakban ezt a módszert szeretnénk használni a nehezebb regularitási problémák vizsgálatára is. (Itt elsősorban kompakt halmazok készítésére gondolunk.)

A dolgozat negyedik fejezete a regularitás számítógépes vizsgálatával foglalkozik, egyrészt a kvázi-lineáris egyenletekkel foglalkozunk, illetve a Jacobi-mátrixok rangjára vonatkozó feltétel geometriai interpretálásával foglalkozunk. Ebben a fejezetben szóba kerül még az előbb említett kompakt halmaz készítés is. A módszerek kidolgozásában a MAPLE (15. verzió) illetve a SAGE (több verzió) komputeralgebra rendszert használtuk. A komputeralgebra rendszerek alkalmazása nem jelent erős informatikai korlátozást, a feladatok döntő része nem különösebben időigényes, emellett az alacsony szintű számítások algoritmikus kérdései helyett azonnal a matematikai problémával foglalkozhatunk.
Maga Aczél János több magyarországi egyetemnek volt a kutatója, valószínűleg ez is oka annak hogy hazánkban jelenleg is komoly kutatások folynak e függvényegyenletek területén. Ezért is található az irodalomjegyzékben igen sok magyar vonatkozású hivatkozás.

1. A SZÁMÍTÓGÉPES ALKALMAZÁSOK ÁTTEKINTÉSE

Ebben a fejezetben áttekintjük, milyen megoldások léteznek az irodalomban, illetve emellett milyen egyszerű módon tudjuk a komputeralgebra rendszereket alkalmazni a függvényegyenletek vizsgálata közben. Ezek nagy része elsősorban nem algoritmikus megközelítés, inkább a rendszerekbe épített precíz szimbolikus számítási mechanizmusokat használjuk, úgy mintha egy nagyon okos kalkulátorral lenne dolgunk.

1.1. Algebrai manipulációk

Az esetek többségében a függvényegyenletek megoldása közben szükségünk van valamilyen speciális helyettesítési értékkel számolni, vagy új változók bevezetésére. Ezek lehetnek természetesen igen triviálisak, de lehetnek trükkösebbek is, amikor jól jön egy kis segítség, kivédendő az elszámolásokat.

1.1.1. Egy egyszerű egyenlet

Tekintsük a következő egyenletet:

$$f(x) + f\left(\frac{1}{1-x}\right) = x\,, \tag{1.1}$$

ahol $x \in \mathbb{R}$ és $x \neq 0, 1$. MAPLE -ben a számolás nagyon egyszerű, mint ahogy az alábbi kódrészletben látni fogjuk. Nyilvánvaló azonban, hogy itt ténylegesen csak okos kalkulátorként használjuk a programot, a lépések természetesen a mi döntésünktől függenek. A következő rövid kódrészletben ezt a számolást mutatjuk meg:

Az $f(x)+f\left(\frac{1}{1-x}\right)=x$ egyenlet megoldása

```
>eq1 := f(x)+f(1/(1-x)) = x;
```

$$eq1 := f(x)+f\left(\frac{1}{1-x}\right)=x$$

```
>eq2 := simplify(subs(x = 1/(1-x), eq1));
```

$$eq2 := f\left(-\frac{1}{-1+x}\right)+f\left(\frac{-1+x}{x}\right)=-\frac{1}{-1+x}$$

```
>eq3 := simplify(subs(x = \dfrac{(-1+x)}{x}, eq1));
```

$$eq3 := f\left(\frac{-1+x}{x}\right)+f(x)=\frac{-1+x}{x}$$

```
>simplify(eq1-eq2+eq3);
```

$$2f(x)=\frac{x^3-x+1}{(-1+x)x}$$

Ezt az egyszerű egyenletet a továbbiakban más módszerrel is megvizsgáljuk.

1.1.2. A 2,2 additivitási egyenlet

A következő egyenlet az információelméletből eredeztethető, az entrópia karakterizálásában van fontos szerepe. A példa Járai [29]-ből származik, ott azonban több sajtóhiba szerepel a megoldásban.

$$\begin{aligned} &f(xy)+f(x(1-y))+f((1-x)y)+f((1-x)(1-y))= \\ &f(x)+f(1-x)+f(y)\,, \end{aligned} \tag{1.2}$$

ahol $0<x,y<1$. A célunk az, hogy olyan alakra hozzuk az egyenletet, hogy a következő fejezetben tárgyalt regularitási tételek alkalmazhatóak legyenek. Ehhez most a $t=xy$ helyettesítésre van szükségünk, majd ki kell fejeznünk $f(t)$-t.

A 2, 2 additivitási egyenlet I

```
>eq1 := f(x*y)+f(x*(1-y))+f((1-x)*y)+f((1-x)*(1-y))
      = f(x)+f(1-x)+f(y)+f(1-y)
>eq2 := f(t) = solve(subs(x = t/y, algsubs(x*y = t, eq1)), f(t))
```

Eredményül az

$$f(t) = f\left(\frac{t}{y}\right) + f\left(1-\frac{t}{y}\right) + f(y) + f(1-y) - f\left(\frac{t(1-y)}{y}\right) - f\left(\left(1-\frac{t}{y}\right)y\right) - f\left(\left(1-\frac{t}{y}\right)(1-y)\right)$$

egyenletet kapjuk. A később ismertetendő regularitási tételek segítségével belátható, hogy ha az f függvény mérhető, úgy C^∞ is.
Szokásos megoldási módszer az, hogy differenciálható függvény esetén annak valamely kifejezését egymásutáni deriválásokkal megpróbáljuk „eltüntetni”. Mindkét változó szerint differenciálva az eredeti egyenletet az eredmény a következő formájú:

$$\begin{aligned} &xy\cdot f''(xy) + f'(xy) - x(1-y)\cdot f''(x(1-y)) - f'(x(1-y)) \\ &- xy(1-y)\cdot f''(y(1-y)) - f'(y(1-y)) \\ &+ (1-x)(1-y)\cdot f''((1-x)(1-y)) - f'((1-x)(1-y)) \ . \end{aligned}$$

Járai [29]-ben ezt az egyenletet részletesen megvizsgálta és megoldotta MAPLE -ben; itt csak arra szeretnénk rámutatni, hogy a fenti egyenletben látható mintát a jelenlegi szimbolikus rendszerek segítségével nem tudjuk közvetlenül kezelni. A benne látható $g(t) = f'(t) + tf''(t)$ háromszor is előforduló mintázatot egyszerű helyettesítésekkel nem lehet átalakítani. Például MAPLE -ben a háttérben, az $f''(xy)$ részkifejezés tárolása a következőképp történik:

```
FUNCTION(3)
   FUNCTION(3)
      FUNCTION(3)
         NAME(4): `@@` #[protected, _syslib]
         EXPSEQ(3)
            NAME(4): D #[protected, _syslib]
            INTPOS(2): 2
      EXPSEQ(2)
         NAME(4): f
   EXPSEQ(2)
      PROD(5)
```

```
NAME(4): x
INTPOS(2): 1
NAME(4): y
INTPOS(2): 1
```

(Ehhez a **dismantle**) függvényt használtuk.) Az ilyen adatszerkezetek rendszeren belüli kezelése - ha megoldható is - túl speciális és az adott programcsomaghoz kötődik, könnyebb, ha konvertáljuk sztringgé és azon végezzük az átalakításokat. A későbbiek során ezt az utat választottuk például a belső függvények kezelésére.

1.2. Differenciálegyenletek megoldása

A függvényegyenletek megoldásának igen hatékony módszere az, hogy átalakítjuk differenciálegyenletté, majd az így keletkező egyenleteket oldjuk meg. A komputeralgebra rendszerek közül talán a MAPLE a „legfelkészültebb”, mind szimbolikus mind numerikus differenciálegyenlet megoldásban.

1.2.1. Az additív Cauchy egyenlet

Ha az egyenletben az ismeretlen f függvény egyszer differenciálható akkor az ismert módon differenciálva az egyenletet y szerint, majd az $y = 0$ helyettesítést végrehajtva meg tudjuk oldani a kapott differenciálegyenletet. Az eljárás SAGE -ben:

```
f = function('f')
y =var('y')
eq1 = f(x+y)==f(x)+f(y); print eq1
eq2=diff(eq1,y); print eq2
eq3 = eq2.subs(y=0); print eq3
f(x) = desolve(eq3,f(x),y); f(x)
```

```
    f(x + y) == f(x) + f(y)
    D[0](f)(x + y) == D[0](f)(y)
    D[0](f)(x) == D[0](f)(0)
    x*D[0](f)(0) + c
```

Megjegyezzük, hogy meglepő módon ezt ilyen egyszerűen MAPLE -ben nem tudjuk megoldani. Ennek oka az, hogy a SAGE függvényfelfogása áll közelebb a matematikai gondolkodáshoz. Amennyiben az egyenletben az ismeretlen f függvény kétszer differenciálható, akkor az ismert módon egymásután differenciálva az egyenletet a két változó szerint, majd az $y = 0$ helyettesítést végrehajtva meg tudjuk oldani a kapott differenciálegyenletet. Ez már MAPLE -ben is könnyű, de továbbra is szükségünk van egy függvénnyé konvertáló **unapply** utasításra.

```
> eq := f(x+y) = f(x)+f(y);
                      f(x + y) = f(x) + f(y)
> d1 := diff(eq, x);
                                  d
                      D(f)(x + y) = --- f(x)
                                  dx
> d2 := diff(d1, y);
                      @@(D, 2)(f)(x + y) = 0
> d3 := subs(y = 0, d2);
                        @@(D, 2)(f)(x) = 0
> sol := dsolve(d3, f(x));
                      f(x) = _C1 x + _C2
> f := unapply(rhs(sol), x);
                      x -> _C1 x + _C2
> solve(eq, _C2);
                                0
```

1.2.2. Az információelmélet alapegyenlete

Tekintsük az

$$f(x) + (1-x)\, f\left(\frac{y}{1-x}\right) = f(y) + (1-y)\, f\left(\frac{x}{1-y}\right) \tag{1.3}$$

egyenletet ahol $0 < x, y, x+y < 1$. A Járai [29]-ben írt manuális megoldás lépésről lépésre megvalósítható MAPLE -ben, ráadásul látszik, mennyivel egyszerűbb a regularitási tétel segítségével levezetett differenciálegyenlet kezelése.

```
> eq := f(x)+(1-x)*f(y/(1-x)) = f(y)+(1-y)*f(x/(1-y));
```

$$eq := f(x) + (1-x)\, f\left(\frac{y}{1-x}\right) = f(y) + (1-y)\, f\left(\frac{x}{1-y}\right)$$

```
> eq2 := diff(eq, x, y);
```

$$eq2 := \left(D^{(2)}\right)(f)\left(\frac{y}{1-x}\right) y\,(1-x)^{-2} = \left(D^{(2)}\right)(f)\left(\frac{x}{1-y}\right) x\,(1-y)^{-2}$$

```
> s1 := g(t) = t*(D(D(f)))(t);
```

$$s1 := g(t) = t\left(D^{(2)}\right)(f)(t)$$

```
> eq3 := g(y/(1-x))/(1-x) = g(x/(1-y))/(1-y);
```

$$eq3 := g\left(\frac{y}{1-x}\right)(1-x)^{-1} = g\left(\frac{x}{1-y}\right)(1-y)^{-1}$$

```
> sol := solve({u = y/(1-x), v = x/(1-y)}, {x, y});
```

$$sol := \left\{x = \frac{v(-1+u)}{uv-1}, y = \frac{u(v-1)}{uv-1}\right\}$$

```
> assign(sol);
> eq4 := simplify(subs(sol, eq3));
```

$$eq4 := \frac{g(u)(uv-1)}{v-1} = \frac{g(v)(uv-1)}{-1+u}$$

```
> eq5 := eq4*denom(lhs(eq4))*denom(rhs(eq4))/(u*v-1);
```

$$eq5 := (-1+u)\, g(u) = (v-1)\, g(v)$$

```
> eq6 := g(t) = c/(1-t);
```

$$eq6 := g(t) = \frac{c}{1-t}$$

```
> de := subs(eq6, s1);
```

$$\frac{c}{1-t} = t\left(D^{(2)}\right)(f)(t)$$

```
> dsol:=dsolve(de, f(t));
```

$$dsol := f(t) = ct\ln(t) + c\ln(-1+t) - c\ln(-1+t)\,t - c + \_C1\,t + \_C2$$

```
> collect(dsol, {c, ln});
```

$$f(t) = (t\ln(t) - 1 + (1-t)\ln(-1+t))\,c + \_C1\,t + \_C2$$

A differenciálegyenlet megoldás 2-3 másodpercet vesz igénybe, az utolsónak alkalmazott **collect** parancs arra szolgál, hogy olvashatóbb formára alakítsuk a megoldást. Hasonlóan tudjuk kezelni az információelmélet általánosított alapegyenletét:

$$f_0(x) + (1-x)^{\alpha} f_1\left(\frac{y}{1-x}\right) = f_3(y) + (1-y)^{\alpha} f_2\left(\frac{x}{1-y}\right). \qquad (1.4)$$

Itt $f_0, f_1, f_2, f_3 :]0,1[\to \mathbb{R}$ az ismeretlenek, $\alpha \in \mathbb{R}$ tetszőleges konstans, továbbá $0 < x + y < 1$. A kétszeri deriválással „megszabadulhatunk” az f_0, f_3 függvényektől. A $tf_i''(t) + (1-\alpha)f'(t), i = 1, 2$ mintát itt is manuálisan kell behelyettesítenünk, innen

ugyanúgy folytathatjuk, mint az (1.3) esetén.

1.2.3. Lineáris függvényegyenlet megoldása számítógéppel

Legyen függvényegyenletünk valós, kétváltozós, lineáris, azaz

$$h_0(x,y)f_0(g_0(x,y)) + \cdots + h_n(x,y)f_n(g_n(x,y)) = F(x,y) \tag{1.5}$$

alakú, ahol $g_0, \ldots, g_n$, $h_0, \ldots, h_n$ és F ismert valós értékű függvények az $\Omega \subset \mathbb{R}^2$ halmazon, az $f_0, \ldots, f_n$ függvények pedig ismeretlenek. A 1 fejezetben ismertetett regularitási tételekből következik az, hogy ha az ismert függvények k-ad rendben differenciálhatók $(1 \leqq k \leqq \infty)$, akkor az ismeretlen függvények mérhetőségéből következik ugyanilyen rendű differenciálhatóságuk. Ezen típusú függvényegyenletek megoldására Páles [37]-ben fejlesztett ki egy módszert, mellyel belátható, hogy ha az ismeretlen függvények analitikusak, akkor az ismeretlen függvény mérhetőségéből következik azok analitikussága. Erre a módszerre alapozva kidolgozott, egy algoritmust, mellyel lépésenként egy kivételével egy differenciáloperátor segítségével kiüthetők az ismeretlen függvények, a megmaradóra pedig egy differenciál-függvényegyenletet kapunk.
Legyen $C^k(\Omega)$ az Ω halmazon értelmezett k-szor folytonosan deriválható valós értékű függvények halmaza. Legyen továbbá $D^k(\Omega)$ a

$$D = \sum_{\substack{i,j \geqq 0 \\ i+j<=k}} \alpha_{ij} \partial_x^i \partial_y^j$$

alakú parciális differenciáloperátorok halmaza, ahol $\alpha_{ij} \in C(\Omega)$. Azt mondjuk, hogy a $D_1, \ldots, D_n \in D^k(\Omega)$ differenciáloperátorok **függetlenek**, ha

$$\sum_{i=1}^{n} \alpha_i(x,y) D_i = 0$$

pontosan akkor teljesül, ha az α_i-k azonosan nullák az Ω halmazon. (Példa: független rendszerre a standard $\{id, \partial_x, \ldots, \partial_x^{k-1}, \partial_y, \partial_y^{k-1}\}$ halmaz). Ha $u_1, \ldots, u_n$ a Ω-n értelmezett $\mathbb{R}^m$-beli értékű folytonos függvények, akkor azt mondjuk, hogy ezen függvényrendszer rangja r, ha létezik olyan $v_1, \ldots, v_r$ részrendszere, hogy valamely $\Omega^* \subset \Omega$ sűrű részhalmaz minden (x,y) pontjában a $(v_1(x,y), \ldots, v_r(x,y))$ mátrix rangja r.

Nyilvánvaló, hogy nem minden $\mathbb{R}^m$-beli értékű függvényrendszernek van rangja, azonban ha a függvények analitikusak, és Ω összefüggő nyílt halmaz, akkor a rang létezik. Ennek bizonyítása Páles [37]-ben található. Ha adott differenciáloperátorok egy $D_1, \ldots, D_m$ rendszere $D^k(\Omega)$-ból, akkor a láncszabály alkalmazásával

$$D_l\left(h_i\left(x,y\right) f_i\left(g_i\left(x,y\right)\right)\right) = \sum_{j=0}^{k} h_{ij}^{l}\left(x,y\right) f_i^{(j)}\left(g_i\left(x,y\right)\right) \;\; (x,y) \in \Omega\,,$$

ahol $0 \leqq i \leqq n, 0 \leqq j \leqq k, 1 \leqq l \leqq m$. Definiáljuk most a következő mátrixokat:

$$\begin{aligned}
&h_{ij} = (h_{ij}^1, \ldots, h_{ij}^m)^T,\; i = 0,1,\ldots,n,\; j = 0,1,\ldots k\,;\\
&H_i = (h_{i0}, \ldots, h_{ik}),\; i = 0,1,\ldots n\,;\\
&H = (H_1, \ldots, H_n)\,;\\
&H^* = (H_0, H_1, \ldots, H_n)\,.
\end{aligned}$$

A fenti jelölések segítségével kimondható az alábbi tétel:

1.2.1. Tétel. *Tegyük fel, hogy $\{D_1, \ldots, D_m\}$ független operátorok és a H mátrixnak, azaz a*

$$\{h_{10}, \ldots, h_{1k}, \ldots, h_{b0}, \ldots, h_{nk}\} \tag{1.6}$$

rendszernek a rangja létezik és legfeljebb $m-1$. Ekkor létezik egy nem azonosan zérus $D \in D^k(\Omega)$ operátor, hogy

$$D\left[h_i\left(x,y\right) f_i\left(g_i\left(x,y\right)\right)\right] = 0, \qquad (x,y) \in \Omega \qquad i = 1,\ldots,n\,. \tag{1.7}$$

Ha még H^*-nak is van rangja és az nagyobb H rangjánál, akkor létezik olyan $D \in D^k(\Omega)$, hogy (1.7) teljesül, de az (1.5) baloldalán első tagjára

$$D\left[h_0\left(x_0,y_0\right) f_0\left(g_0\left(x_0,y_0\right)\right)\right] \neq 0$$

valamilyen $f_0 \in C^k(\Omega)$ és $(x_0,y_0) \in \Omega$ esetén.
Ha $v_1 = (v_{11}, \ldots, v_{1m})^T, \ldots, v_r = (v_{r1}, \ldots, v_{rm})^T$ generátorrendszere az (1.6)-nek,

akkor a D operátor a következő formában írható:

$$D = \det \begin{vmatrix} v_{11}(x,y) & \cdots & v_{(m-1)1}(x,y) & D_1 \\ \vdots & \ddots & \vdots & \vdots \\ v_{1m}(x,y) & \dots & v_{(m-1)m}(x,y) & D_m \end{vmatrix},$$

ahol a $v_{ij}, r < i < m, 1 \leqq j \leqq m$ együtthatók tetszőlegesen választatók.

A bizonyítást lásd Páles [37]. A tétel alapján a következő algoritmus építhető fel:

(A) Input adatok: $n, h_0, \dots, h_n, g_0, \dots, g_n, F$;

(B) $k \leftarrow 0$;

(C) $k \leftarrow k + 1$;

(D) $m \leftarrow m(m+1)/2, D = \{id, \partial_x, \dots, \partial_x^{k-1}, \partial_y, \dots, \partial_y^{k-1}\}$;

(E) A $H_0, H_1, \dots, H_n$ mátrixok meghatározása;

(F) H és H^* rangjának meghatározása;

(G) Ha $m - 1 < rank(H)$, akkor (C);

(H) Ha $rank(H) = rank(H^*)$, akkor (C);

(I) D meghatározása;

(J) Differenciál-függvényegyenlet felállítása f_0-ra.

Az eljárást Házy Attila fejlesztette tovább Házy [25]-ban, amelyben az eredményül kapott k-ad rendű lineáris differenciál-függvényegyenletet lépésenként közönséges differenciálegyenletté alakítja.

1.3. Sorfejtések

1.3.1. Az additív Cauchy egyenlet

Ha feltételezzük, hogy az f ismeretlen függvény analitikus, akkor az origó körüli sorfejtéssel meg tudjuk oldani az egyenletet.

```
>eq0 := f(x+y) = f(x)+f(y):
>eq1 := subs(y = x, eq0):
>p1 := convert(taylor(lhs(eq1), x), polynom):
>p2 := convert(taylor(rhs(eq1), x), polynom):
>P := p1-p2:
```

$$P := -f(0) + D^{2)}(f)(0) * x^2 + D^{(3)}(f)(0) * x^3 + \frac{7}{12} * D^{(4)}(f)(0) * x^4 + \frac{1}{4} * D^{(5)}(f)(0) * x^5$$

Innen leolvasható a megoldás: az elsőfokú együttható tetszőleges lehet, mivel kiesik a kivonással, viszont az egyenlőség miatt az összes többinek zérusnak kell lennie, tehát $f(x) = cx$.

1.3.2. Közepek invarianciaegyenlete

A közepek és a függvényegyenletek kapcsolatának a referenciában felsoroltakon kívül is igen szerteágazó irodalma van. Itt csak a Baják, Páles szerzőpáros Baják and Páles [7, 8, 9] cikkeit említjük meg, melyben a MAPLE komputeralgebra rendszert használják Taylor-sorfejtés végrehajtására. **Középen** egy olyan $M : \mathbb{R}_+^2 \to \mathbb{R}_+$ függvényt értünk, melyre $x, y \in \mathbb{R}_+$ esetén

$$\min\{x, y\} \leqq M(x, y) \leqq \max\{x, y\}$$

teljesül. Ha az előző formulában csak $x = y$ esetén áll fenn egyenlőség, úgy a közepet **szigorúnak** nevezzük. Az M közép **homogén**, ha $M(\lambda x, \lambda y) = \lambda M(x, y)$. Ha $x_1, y_1 \in \mathbb{R}_+$ adott kezdőértékek, akkor az

$$x_{n+1} = M(x_n, y_n),\ y_{n+1} = N(x_n, y_n),\ \text{ahol}\ (n \in \mathbb{N})$$

iterációt **Gauss-iterációnak** nevezzük, ez a jól ismert aritmetikai – geometria közép iteráció általánosítása. Ismert, hogy összehasonlítható, szigorú közepek esetében a fenti két sorozat minden kezdőértékre közös határértékhez konvergál, [lásd Borwein és Borwein 14]. A kapott határérték egy közepet definiál, ez a két közép $K = M \otimes N$ **Gauss kompozíciója**. Az így definiált közép a következő invariancia egyenlettel karakterizálható:

$$K\left(M\left(x, y\right), N\left(x, y\right)\right) = K\left(x, y\right)\ \ x, y \in \mathbb{R}_+ .$$

A fent említett cikkek azt az esetet vizsgálják, amikor a fenti egyenletben a közepek Gini- illetve Stolarsky-közepek. A megoldások erősen támaszkodnak a homogenitásra, illetve a Gini-közepek esetén azok szimmetrikus voltára. A számítógép segítsége lényegében Taylor-együtthatók kiszámítását és bizonyos algebrai manipulációkat jelent. A számítást tartalmazó viszonylag hosszabb MAPLE -kód a mellékletben található: (); a számítások az eredeti cikkből vannak. A páratlan rendű Taylor-sorfejtés eredménye zérus, a kód páros rendűeket számolja 2 és 12 rend között. A számolás eredménye egy hatváltozós 11-ed fokú polinomiális egyenletrendszer lenne, ennek megoldása nem túl sok reménnyel kecsegtet. Az $sb1 := a = w + v + \sqrt{r+s}$; alakú helyettesítésekkel egy változó kiküszöbölhető. A cikk a Taylor-sorok kiszámítása után rezultáns számításával keresi meg a megoldást. Az itt közölt kódban kiszámoltuk a Taylor-polinomok Gröbner-bázisát, ami némileg lerövidíti az egyenletrendszer megoldását.

1.4. Általános megoldó

A mai komputeralgebra rendszerek az egyenletek rendkívül széles skáláját képesek megoldani: a polinomiális-, transzcendens-, differenciál-, diofantikus egyenleteket, kongruenciákat, rekurzív összefüggéseket. Ezek mögött jól kidolgozott elméleti és algoritmikus metódusok állnak, a függvényegyenletekre ez azonban távolról sem mondható el, nincs is olyan programcsomag, ami függvényegyenleteket tudna megoldani. Ennek több oka is van. Nincs egységes metodológia a függvényegyenletek megoldására, az alkalmazott módszerek jelentős része heurisztikus. A problémakör igen nehéz, a megoldások erősen függenek a konkrét egyenlet értelmezési tartományától, emellett a diszciplína sem igazán ismert más területek kutatói számára. A szakirodalomban egyetlen olyan kísérletet ismerünk, amely megpróbált ezzel a problémakörrel átfogóan foglalkozni. Először ezt ismertetjük, majd vázolunk néhány más lehetséges megközelítést megoldóprogram(ok) felépítéséhez.

1.4.1. Castillo és Iglesias programcsomagja

Castillo és Iglesias 1995-ben a „Mathematics with Vision: Proceedings of the First International Mathematica Symposium” konferencián mutatták be, később cikk formájában is ismertették egy a MATHEMATICA komputeralgebra rendszerben írott prog-

ramcsomagjukat, lásd [4, 17]. A célkitűzésük igen ambíciózus, a következőképp foglalható össze:

- kidolgozni egy adatbázist, mely tartalmazza a függvényegyenleteket, különböző tartományokat és a hozzájuk tartalmazó megoldást;
- olyan rendszert kidolgozni, amely hatékonyan használja ezt az adatbázis;
- néhány algoritmust implementálni függvényegyenletek megoldására.

Ez a kísérlet ma már csak történelmi érdekességű, a MATHEMATICA internetes archívumában lelhető fel némi információ róla, a rendszerbe való integrálása nem történt meg. A kitűzött célok csak részben valósultak meg, a forráskód elemzése alapján azt mondhatjuk, az „adatbázis” és az „algoritmusok” fixen be vannak égetve a kódba.

1.4.2. Megjegyzés

Nagy biztonsággal mondhatjuk, hogy a függvényegyenletekre univerzális megoldó algoritmus nem létezik. Felsorolunk néhány okot, mely szerintünk ezt lehetetlenné teszi.

- Gyakran igen „egyszerű” és „szép” függvényegyenletek „vad” függvények is lehetnek a megoldásai, ilyen például a következő fejezetben tárgyalt általános megoldása a Cauchy additív egyenletnek.
- Gyakran úgy oldunk meg egyenleteket, hogy egy egyenletről belátjuk, ekvivalens a másikkal (azaz ugyanazok a függvények a megoldásaik), az ekvivalencia bizonyítása azonban gyakran ugyanolyan bonyolult, mint a megoldás. Példa ekvivalens függvényegyenletekre: a Cauchy additív egyenlet ekvivalens az $f(x+y+xy) = f(x) + f(x) + f(xy)$ egyenlettel.
- A megoldásokat mindig ellenőriznünk kell, mivel nincs túl sok eszközünk kizáró kritérium felállítására. Példaképpen a $2f(x+y)+6y^3 = f(x+2y)+x^3, \forall x, y \in \mathbb{R}$ egyenletbe $y = 0$-t helyettesítve kapjuk, hogy $f(x) = x^3$, de könnyen meggyőződhetünk róla, hogy ez nem elégíti ki az eredeti egyenletet.

- Gyakran a megoldás tetszőleges konstansok vagy tetszőleges függvények bevezetését igényli. Ha tekintjük például az $f(x, f(y, z)) = f(f(x, y), z)$ asszociativitási egyenletet, ennek általános megoldása $f(x, y) = g^{-1}(g(x) + g(y))$, ahol g tetszőleges invertálható függvény.

Bizonyos területeket azonban természetesen erőteljesen lehet támogatni számítógépes megoldásokkal.

1.5. Megoldás helyettesítések véges csoportjával

Térjünk vissza a 1.1 egyenletre. Ebben az f ismeretlen függvényt a következő belső függvényeken hajtjuk végre:

$$g_0(x) = x,\ g_1(x) = \frac{1}{1-x}\,,$$

és $x \in \mathbb{R}$ és $x \neq 0, 1$. Könnyen ellenőrizhető, hogy $g_1^2(x) = (g_1 \circ g_1)(x) = \frac{x-1}{x}$ és $g_1^3(x) = g_0(x) = x$. Vagyis az g_0, g_1, g_1^2 függvények a kompozíció műveletére csoportot alkotnak, itt g_0 az egységelem. A példa megoldásában rendre az x-et helyettesítettük a $g_1(x), g_1^2(x)$ kifejezésekkel, majd a keletkező egyenletrendszert $f(x)$-re megoldottuk. A problémát általánosabban is megfogalmazhatjuk: ha a $G = \{g_1(t), g_2(t), \ldots, g_n(t)\}$ leképezések a kompozíció műveletére csoportot alkotnak, akkor a

$$\sum_{i=1}^{n} a_i(t) f(g_i(t)) = h(t)$$

függvényegyenlet, ahol a_i és b ismert függvények könnyen megoldható, ha t helyére rendre behelyettesítjük $g_i(t)$-t, majd a keletkező lineáris egyenletrendszert megoldjuk. Az ezen típusú, vagy általánosabban, az

$$F(f \circ g_1(t), \ldots, f \circ g_n(t), t) = h(t)$$

alakú egyenletek megoldhatóságát részletesen tárgyalja Besssenyei szerzőtársaival a [10, 11, 12, 13] cikkekben. Mi itt most azt vizsgáljuk meg, hogyan lehet a fenti típusú egyenleteket komputeralgebra rendszerek segítségével megoldani.

1.5.1. A probléma megfogalmazása

Tekintsük a

$$F(f \circ g_1(x), \ldots, f \circ g_k(x)) = h(x)$$

függvényegyenletet, ahol a $g_1, \ldots, g_k$ belső függvények véges csoportot alkotnak illetve generálnak a függvénykompozíció műveletével. F és h ismertek, minden függvény a $H \subset \mathbb{R}$ halmazt képezi $\mathbb{R}$-be. Feltételezve F linearitását, egyenletünk a következő alakú lesz

$$\alpha_1(x) \cdot f((g_1(x)) + \ldots + \alpha_k(x) \cdot f(g(_k(x))) = h(x) \ (k \in \mathbb{N}) \,. \tag{1.8}$$

Ezen típusú függvényegyenletekkel már Babagge is foglalkozott munkáiban. Könnyen kezelhetősége miatt nagy népszerűségnek örvend a matematika versenyeken is, a Brodskii and Slipenko [15], Small [38] könyvek egy-egy fejezetet szenteltek ennek a témakörnek. Az itt kifejtett módszerben a lineáris egyenletekre korlátozódás nem jelent erős megszorítást, a különbség két ponton jelentkezik. Először is a függvényegyenletet megoldhatósági feltételek más a nemlineáris esetben. Ennek következtében az egyenletrendszer felállítása nem lesz olyan egyszerű, az eddig tárgyalt esetben. Másrészt, mivel ekkor nemlineáris egyenletrendszert kapunk, annak megoldása más technikákat igényel.

1.5.2. A háttér

A továbbiakban tekintsük az (1.8) egyenletet, ahol f az ismeretlen függvény,

$$g_1, \ldots, g_k, h \text{ és } \alpha_1, \ldots, \alpha_n$$

az ismert függvények, melyek $\mathbb{R}$ valamely részhalmazát $\mathbb{R}$-be képezik. Tegyük fel továbbá fel, hogy a g_i függvények csoportot alkotnak a függvénykompozíció műveletével. Tekintsük át először a manuális megoldás menetét ebben az általánosabb formában.

1.5.2.1. A manuális megoldás

Legyen $G = \{g_1, \ldots, g_n\}$ a belső függvények csoportja. Készítsük el az egyenlet kompozícióját az összes csoportelemmel! Ekkor a következő egyenletrendszert kapjuk:

$$\alpha_1 \cdot f \circ g_1 \circ g_i + \ldots \alpha_n \cdot f \circ g_k \circ g_i = h \circ g_i \,, \tag{1.9}$$

ahol $1 \leqq i \leqq n$. Természetesen, ha $k < n$, azaz a csoport elemszáma nagyobb, mint a szereplő belső függvények száma, akkor minden $k < i \leqq n$ esetén $\alpha_i \equiv 0$.
$(G, \circ)$ csoport, ezért minden $1 \leqq i, j \leqq n$ esetén van olyan l, hogy $g_i \circ g_j = g_l$, tehát a fenti rendszer jól definiált. A fenti behelyettesítésekkel tehát kaptuk egy lineáris egyenletrendszert, melyben az ismeretlenek az $f \circ g_i$ függvénykompozíciók.
Az előzőekben kifejtett lépések lényegében egy algoritmust vázolnak ezen típusú függvényegyenletek megoldására. Ezért természetesen adódik komputeres módszer alkalmazása a megoldásra. A kapott lineáris egyenletrendszer a komputeralgebra rendszerek mindegyikében jól kezelhető, mind a lineáris egyenletek megoldására, mind a mátrix- és determináns műveletek támogatására hatékony és magas szintű algoritmusok állnak rendelkezésre.
A következőkben áttekintünk néhány csoportelméleti fogalmat, melyekre szükségünk van az algoritmus kidolgozásához.

1.5.2.2. Egy kis csoportelmélet

Legyen $\mathbb{N}_n = \{1, 2, \ldots, n\}$ az első n pozitív egész szám halmaza. Legyen továbbá $H \subset \mathbb{R}$ és $G = \{g_i : i \in \mathbb{N}_n\}$ függvények egy véges halmaza, melyek H-t $\mathbb{R}$-be képezik. tegyük fel, hogy a g_i függvények csoportot alkotnak a függvénykompozícióra. Jelölje $G(H) = (G, \circ)$-vel ezt a csoportot. Az $\mathbb{N}_n$ halmazon definiáljuk a $*$ műveletet a következőképp:

$$i * j = k \text{ pontosan akkor, ha } g_i \circ g_j = g_k.$$

Ekkor $(\mathbb{N}_n, *)$ egy véges permutációcsoport, mely izomorf $G(H)$-vel. Ez egy hű reprezentációja $G(H)$-nek, a fenti műveleti szabály definiálja a permutációcsoport hatását a függvényhalmazon. Jelölje σ_i a permutációcsoport azon elemét, mely a $*i$ hatása a G halmazon, legyen továbbá $\pi_i = \sigma_i^{-1}$.

Ebből a definícióból következik, hogy az (1.9) egyenletrendszer egyenletei nem mások, mint a π_i-k hatásának eredményei az eredeti (1.8) egyenleten. Így a csoport elemeit ismerve az egyenletrendszer előállításához nincsen szükség a kompozíciók végrehajtására, elegendő a permutációkat végrehajtani.
Természetesen merül fel a kérdést, az (1.9) egyenlet milyen feltételek teljesülése mellett oldható meg? Erre a kérdésre a válasz a következő tétel:

1.5.1. Tétel. *Legyen $H \subset \mathbb{R}$ egy nem üres halmaz, $\alpha_1, \ldots, \alpha_n$ és $g_1, \ldots, g_n$ pedig H-t $\mathbb{R}$-be képező függvények és a g_i-k csoportot alkotnak a függvénykompozícióra. Tegyük fel, hogy a*

$$D = \begin{vmatrix} \alpha_{\pi_1}(1) \circ g_1 & \cdots & \alpha_{\pi_1}(n) \circ g_1 \\ \vdots & \ddots & \vdots \\ \alpha_{\pi_1}(n) \circ g_n & \cdots & \alpha_{\pi_1}(n) \circ g_n \end{vmatrix}$$

determináns nem tűnik el H-n. Ekkor egyértelműen létezik olyan $f : H \to \mathbb{R}$ függvény, amelyik megoldása az (1.8) függvényegyenleteknek.

A tétel bizonyítása megtalálható Bessenyei and Kézi [12] cikkében. A [10, 11, 13] cikkekben találhatunk további eredményeket a nemlineáris eset megoldhatóságára, ezek valamilyen analitikus „regularitás” (differenciálhatóság, folytonosan differenciálhatóság, stb.) meglétét tételezik fel a függvényekre, valamint intenzíven használják a kombinatorikus csoportelmélet eszköztárát.

1.5.3. A számítógépes megoldás

Az implementáció a SAGE komputeralgebra rendszerben készült. Mindegyik alkalmazott rendszerről elmondhatjuk, hogy a csoportok kezelésé vagy permutációcsoportok segítségével, vagy mátrixcsoportokkal történik. Az a kérdés azonban megoldatlan, hogyan építünk fel egy halmazból és az elemei közt értelmezett műveletből egy csoportot.

1.5.3.1. Az implementáció lépései

A komputeralgebra rendszerek függvény kezelése nem teljesen azonos a matematikai függvényfogalommal, mivel a matematikai fogalommal szemben inkább a szabály

alapú megközelítést használják. Emellett többféle megadási mód is a rendelkezésünkre áll. A SAGE -ben a következők a lehetőségek:

- A `def` kulcsszóval definiált függvények. Ez a programozási nyelvekből jól ismert általános függvény- illetve procedúra definiálási módszer. Fontos különbség azonban az, hogy az így definiált függvénnyel most tudunk szimbolikus kifejezéseket is kezelni.

- A `lambda` függvények. Ezeket többnyire rövid, in-line definíciókra használjuk, gyakran nevet sem adunk nekik. A technikát a funkcionális nyelvekből kölcsönözték, alapja pedig — amint a nevéből is látszik — a lambda kalkulus.

- Definiált függvények A SAGE -ben a `function` szóval a változót deklarálnunk kell. Ez a megadási módszer nem igényel kódolást, többnyire így lehet értékadás segítségével a beépített matematikai függvényeket más néven eltárolni.

Céljainknak a harmadik módszer felel meg leginkább, mivel viszonylag könnyen kezelhetővé teszi a szimbolikus függvényeket. Megjegyezzük, hogy a függvényegyenlet előfeldolgozása során — ezen a belső függvény kinyerését értjük — az egyenletet egyszerű szöveggé konvertált formában használjuk.
A realizáció a következő lépésekből áll:

1. Technikai előkészítés, vagyis a belső függvények kinyerése az egyenletből.

2. A csoport megkonstruálása és/vagy annak ellenőrzése, hogy a belső függvények csoportot alkotnak-e.

3. A létrehozott csoporttal izomorf permutációcsoport létrehozása.

4. Az egyenletrendszer megkonstruálása.

5. Az egyenletrendszer megoldása.

1.5.4. A csoport létrehozása

Az a feladatunk, hogy a megadott $g_1, \ldots, g_k$ függvényekről eldöntsük, csoportot alkotnak-e, vagy újabb elemekkel ki kell egészítenünk. Megjegyezzük, hogy ez a két kérdés lényegében ugyanaz: mindenképpen képezni kell az elemek szorzatát, majd

keresni az alaphalmazban. Így azt az utat választjuk, hogy a megadott elemekből felépítjük a csoportot. A naiv módszer azonnal adódik: képezzük a halmaz összes párját, szorozzuk össze őket, majd keressük meg a halmazban. Ez azonban nem elég okos megoldás, ráadásul már viszonylag kis k-k esetén is költséges, k^2 szorzást és k^2 keresést igényel. Ehelyett a Dimino-algoritmust használjuk, ami a naiv módszer nyilvánvaló javítása, felhasználjuk közben az elemi csoportelméleti ismereteket.

Egyszerű Dimino Algoritmus Nyilvánvaló, hogy a csoportnak kell egységelemet tartalmaznia, ezért legelső lépésként ezt mindenképpen belerakjuk a a csoportba. A következő lépésben a legelső nem redundáns generátor elem által generált ciklikus részcsoporttal bővítjük a készülő csoportot. Ezek után indukcióval haladunk, minden lépésben a következő nem-redundáns generátor által képezhető mellékosztályokkal bővítünk, ez jelentős keresési időt fog megtakarítani. Egy kicsit részletesebben leírjuk ezt az induktív lépést:
Legyen $S = \{s_1, s_2, \ldots, s_k\}$ a generátorok halmaza. Az S halmaz részhalmazaira vezessük be a $S_0 = \emptyset$ és $1 \leqq i \leqq k$ esetén $S_i = \{s_1, s_2, \ldots, s_i\}$ jelöléseket. Legyen továbbá $H_i = \langle S_i \rangle$ az S_i által generált részcsoport. Az algoritmus először létrehozza a $H_0 = \{e\}$ triviális részcsoportot, ahol e az egységelem. Feltéve, hogy s_1 nem az egységelem, generálja a $H_1 = \{e, s_1, s_1^2, \ldots, s_1^{n_1-1}\}$ részcsoportot, ahol n_1 az s_1 elem rendje. Az első induktív lépésben bővítjük a csoportot a $H_1 \cdot s_2$ mellékosztállyal, azaz az $\{e, s_1, s_1^2, \ldots, s_2, s_1 s_2, s_1^2 s_2, \ldots, s_1^{n_1-1} s_2\}$ halmazzal.(Itt feltettük, hogy $s_2 \notin H_1$.) Ezután megkeresi a további mellékosztály-reprezentánsokat, amik a $g \cdot s$ elemek, ahol $g \in H_1 \cdot s_2$ és $s \in S_2$, majd a megfelelő mellékosztályokat is hozzáadja az eddig elkészült halmazhoz. Majd ezt folytatja az S még hátralévő elemeivel.
Az algoritmus SAGE kódját adjuk meg a (Dimino Algoritmus) programlistában. Az algoritmust lásd például Butler [16]-ben.

A legjobb futási időt akkor érjük el, amikor két lépésben kell a csoportbővítést elvégezni, ekkor $1.5 \cdot k - 2$ szorzásra és k keresésre van szükség.
Ezt az algoritmust lehet javítani, ha minden egyes lépésben ellenőrizzük, hogy a létrehozott H részcsoport normális részcsoport-e $\langle H, s \rangle$ részcsoportban, ahol s a következő nem-redundáns generátor elem. A normalitás ellenőrzése úgy történik, hogy minden $h \in H$-ra megnézzük, hogy $s^{-1}hs \in H$ teljesül-e.

A permutációcsoport elkészítése A következő lépésben a kapott csoportot kell permutációkká „konvertálni”. Ezt két egyszerű lépésben tesszük meg, némi reprezentációelmélet felhasználásával. Az eljárás ismertetése előtt összefoglaljuk a reprezentációelméletből szükséges tudnivalókat.
Legyen G egy véges csoport, $g \in G$ egy eleme. A g egy **n-dimenziós mátrixreprezentációján** egy $M(g)$ lineáris transzformációt értünk, mely egy n-dimenziós V^n vektoreret önmagába képez. Nyilvánvaló, ha rögzítünk egy bázist V^n-ben, akkor ismerjük a reprezentáció mátrixát. A **reguláris reprezentáció** fogalma szoros kapcsolatban áll a g elemnek a csoporton való hatásával: ebben az esetben a vektortér dimenziója megegyezik a csoport rendjével.
Legyen $e_i, i = 1, \ldots, n$ egy ortonormált bázis V^n-ben. Ekkor a reguláris reprezentációt a következőképp tudjuk megadni:

$$M(g_k)e_j = \sum_{i=1}^{|G|} \delta(g_k g_j, g_i)e_i \,,$$

ahol

$$\delta(g_k g_j, g_i) = \begin{cases} 1 & \text{if } g_k g_j = g_i \\ 0 & \text{if } g_k g_j \neq g_i \end{cases}$$

a „Kronecker-delta” függvény. Ennek kódolása a fenti definíció alapján nagyon könnyű. Második lépésünk a permutációk előállítása a mátrix-reprezentációból.
A megvalósítás képes a permutációkat mind a kétsoros alakban, mind a ciklikus formában előállítani, alapértelmezett a kétsoros. Ez ugyanis a mátrix segítségével igen könnyű, mivel a mátrix minden sora pontosan egy darab egyest tartalmaz, a többi zérus. Így az i-edik sorban a megkeressük azt a j.-edik oszlopot, ahol ez az egyes van. A ciklikus permutáció generálása sem nehéz, az első sorban megkeressük az egyetlen egyes j oszlopindexét, majd a j-edik sorban folytatjuk és így tovább. Lásd a (Csoportreprezentációk) SAGE -kódot.

Megjegyezzük, hogy a ciklikus forma alkalmas arra, hogy a csoport további tulajdonságait megvizsgáljuk, mivel a komputeralgebra rendszerekben ezek segítségével tudunk csoportokat definiálni.

Az egyenletrendszer konstruálása és megoldása Ez leginkább a használt rendszer képességeihez kötődő technikai jellegű feladat, nagymértékben függ a szimbolikus képességektől. Összehasonlítva a MAPLE hasonló képességeivel a SAGE ugyan ebben nem túl erős, de mostani igényünknek megfelelő a tudása. Két szimbolikus helyettesítési lépést használunk, először a csoportelemeket helyettesítjük egy ciklusban egyszerű változókkal, majd újabb változókkal a következő ciklusban az $f(\ldots)$ alakú részkifejezéseket. Egyetlen apró technikai problémát kell megoldani, változó számú szimbolikus változó kell generálnunk a SAGE -ben. Ennek egy megoldása a (Változógenerátor) programlistában található. (A megoldás az [1] AskSage honlap alapján.)

1.1 Program: Változógenerátor

```
class VariableGenerator(object):
    def __init__(self, prefix):
        self.__prefix = prefix
    @cached_method
    def __getitem__(self, key):
        return SR.var("%s%s"%(self.__prefix,key))
    g = VariableGenerator('g')
```

Egy példa ha az

```
x^2*f(x) + f((x - 1)/x) == x^2,
```

egyenletet akarjuk megoldani, akkor a generált egyenletrendszer

```
X0*g0^2 + X1 == g0^2
X2*g2^2 + X0 == g2^2
X1*g1^2 + X2 == g1^2
```

Most már használhatjuk a SAGE beépített megoldóját,ami a következő megoldást produkálja:

```
[[X0 == (g0^2*g1^2*g2^2 - (g1^2 - 1)*g2^2)/(g0^2*g1^2*g2^2 + 1),
```

```
X1 == ((g1^2 - 1)*g2^2 + 1)*g0^2/(g0^2*g1^2*g2^2 + 1),
X2 == ((g1^2*g2^2 - g1^2)*g0^2 + g1^2)/(g0^2*g1^2*g2^2 + 1)]].
```

(Megoldhatnánk a Cramer-szabály segítségével is a feladatot)
Annyi dolgunk maradt, hogy a megoldást olvashatóbb formára hozzuk, ezt az eredmények visszahelyettesítésével végezhetjük, utána ellenőrízzük a megoldást.

Néhány megjegyzés Ezt a részt néhány olyan megjegyzéssel zárjuk, melyek a más komputeralgebra rendszerben való implementálást, a megoldhatóság ellenőrzését, és az általánosíthatóságot érintik.

Más komputeralgebra rendszerek Az egyenletrendszer generálását kivéve az eljárás lényegében független — a programnyelvi differenciáktól eltekintve — az alkalmazott CAS-tól, ez is inkább csak technikai jellegű kérdés. Mindegyik komputeralgebra rendszer tartalmaz magas szintű lista- és mátrixkezelési műveleteket, hasonló érvényes a csoportok kezelésére is.

A megoldhatóság ellenőrzése A lineáris esetben nem foglalkoztunk a megoldhatóság ellenőrzésével, a determináns kiszámítása egyenértékű a megoldással, amit megtesz helyettünk a rendszer megoldója. A nemlineáris esetben ez a lépés azonban nem hagyható el. Ezekkel kapcsolatban a [10, 11, 13] cikkében találunk „regularitási” tételeket, melyeket feltételei vizsgálhatók komputerrel, ugyanis ezek Jacobi-determinánsok vizsgálatát jelentik. Ennek következménye újabb nemlineáris egyenlet megoldása lehet.

Általánosíthatóság A nem-lineáris eset annyiban tér el a most vizsgálttól, hogy egyrészről invertálnunk kell a külső függvényt, másrészről viszont a kapott egyenletrendszer már nem lesz lineáris, ez azonban lényegesen nehezebb kérdés, itt a SAGE tudása már elégtelennek bizonyulhat (szemben a MAPLE -el).

2. A REGULARITÁS PROBLÉMAKÖRE

2.1. A nem-kompozit egyenletek regularitási problémája

2.1.1. A probléma megfogalmazása

A függvényegyenletek megoldásának szokásos módszere az, hogy az egyenletet olyan alakra hozzuk, ami az analízis eszközeivel már könnyebben kezelhető, vagyis megpróbáljuk differenciálegyenletté, parciális differenciálegyenletté vagy integrálegyenletté alakítani. Ehhez az átalakításhoz azonban bizonyos „erős" regularitási tulajdonságokat kell belátnunk az ismeretlen függvény(ek)re, amit az ismert függvények tulajdonságaiból és az ismeretlenre feltételezett „gyenge" regularitási tulajdonságokból tudunk levezetni. Gyakori az a szituáció, amikor egy függvényegyenlet „szép" reguláris megoldással rendelkezik, míg általános megoldása igen különös függvény is lehet. A legközismertebb példa erre az

$$f(x+y) = f(x) + f(y)$$

additív Cauchy egyenlet. Ennek már igen gyenge analitikus feltételek (mérhetőség) mellett is $f(x) = cx, c \in \mathbb{R}$ a megoldása, ami C^∞, tehát erősen reguláris. Az általános megoldásokhoz tekintsük $\mathbb{R}$-et mint $\mathbb{Q}$ feletti vektorteret. A kiválasztási axióma segítségével belátható, hogy ennek van bázisa, amit Hamel-bázisnak nevezünk. Ha $H \subset \mathbb{R}$ Hamel–bázis, akkor $\forall x \in \mathbb{R}$-re teljesül, hogy $\exists r_1, r_2, \ldots, r_n \in \mathbb{Q}$, amikkel $x = \sum_{i=1}^{n} r_i h_i, h_i \in H$. Az additív Cauchy egyenlet általános megoldását úgy kapjuk, hogy egy H Hamel–bázison tetszőlegesen definiáljuk az f függvényt, majd $\forall x \in \mathbb{R}$–re:

$$f(x) = \sum_{i=1}^{n} r_i x_i, x_i \in H\,.$$

A nemlineáris megoldások talán legmeglepőbb tulajdonsága az, hogy az

$$\{(x, f(x))\, x \in \mathbb{R}\}$$

halmaz sűrű $\mathbb{R}^2$-ben.
Jelen dolgozatban a következő speciális mégis elegendően általános formájú függvényegyenletek regularitási kérdését vizsgáljuk:

$$f(x) = h\Big(x, y, f\big(g_1(x,y)\big), \ldots, f\big(g_n(x,y)\big)\Big), \tag{2.1}$$

ahol $h, g_1, \ldots, g_n$ az ismert függvények, f az ismeretlen függvény, x, y pedig valamilyen (nem feltétlenül azonos) véges dimenziós euklideszi térből valók. Az erős regularitási tulajdonságok bizonyítása többlépcsős folyamat:

(I) a mérhetőségből következik a folytonosság;

(II) a Baire tulajdonságból következik a folytonosság;

(III) a folytonosságból következik a majdnem mindenhol differenciálhatóság;

(IV) a majdnem mindenhol differenciálhatóságból következik a folytonosan differenciálhatóság;

(V) a p-szer differenciálhatóságból következik a $p+1$-szer differenciálhatóság;

(VI) a végtelen sokszor differenciálhatóságból következik az analitikusság.

A fentiek alapján regularitási problémákról kell beszélnünk. Járai [29] alapján

2.1.1. Probléma (Alapprobléma)**.** *Legyenek X, Y és Z megfelelően az $\mathbb{R}^r, \mathbb{R}^s$, és $\mathbb{R}^t$ nyílt részhalmazai (r, s, t természetes számok), és legyen D az $X \times Y$ nyílt részhalmaza, W pedig $D \times Z^n$ nyílt részhalmaz.*
Legyenek $f : X \to Z, g_i : D \to X$ $(i = 1, 2, \ldots, n)$, és $h : W \to Z$ függvények. Tegyük fel, hogy

1. *ha $(x, y) \in D$, akkor $(x, y, f(g_1(x, y), \ldots, f(g_n(x, y))) \in W$ és*

$$f(x) = h\Big(x, y, f\big(g_1(x,y)\big), \ldots, f\big(g_n(x,y)\big)\Big), n \in \mathbb{N}\,;$$

2. *h analitikus;*

3. *g_i analitikus és minden $x \in X$ esetén van olyan y, melyre $(x, y) \in D$ és $\frac{\partial g_i}{\partial y}(x, y)$ rangja r $(i = 1, 2, \ldots, n)$.*

Igaz-e, hogy minden olyan f,mely mérhető (vagy rendelkezik a Baire-tulajdonsággal) analitikus?

A továbbiakban is $f, h,$ és g fogja jelölni az ismeretlen függvény(eke)t, a külső függvény(eke)t és a belső függvény(eke)t. A fenti problémának számtalan variánsa létezik, melyekben a regularitási feltételek változnak, például először mérhetőséget, folytonosságot, monotonitást tételezünk fel vagy C^p simaságot tételezünk fel valamely $p \in \mathbb{N}$ vagy $p = +\infty$ vagy $p \in \omega$ esetén és a folytonosságot, differenciálhatóságot, C^∞ differenciálhatóságot, analitikusságot várunk el.

2.1.2. Eredmények

A teljes válasz a fenti problémára nem ismert. A fenti hat pontnak megfelelően lehet a regularitás részkérdéseit vizsgálni. Járai [29]-ben (I),(II),(IV) és (V)-re a válasz teljes körű, a (III) és (VI) kérdésekre további feltételek mellett létezik válasz. A következőkben bizonyítás nélkül összefoglaljuk ezeket az eredményeket.

2.1.1. Tétel. *A (2.1.1) jelöléseivel ha h folytonos és a g_i függvények folytonosan differenciálhatók, akkor minden Lebesgue mérhető vagy Baire-tulajdonságú f megoldás egyben folytonos is.*

2.1.2. Tétel. *A (2.1.1) jelöléseivel ha a h és a g_i függvények p-szer folytonosan differenciálhatók, akkor minden majdnem mindenütt differenciálható f megoldás egyben p-szer folytonosan differenciálható ($1 \leq p \leq \infty$).*

2.1.3. Tétel. *A (2.1.1) jelöléseivel ha a h és a g_i függvények p-szer folytonosan differenciálhatók, akkor minden lokálisan korlátos változású f megoldás egyben p-szer folytonosan differenciálható ($1 \leq p \leq \infty$).*

2.1.4. Tétel. *A (2.1.1) jelöléseivel ha a h és a g_i függvények $\max\{2, p\}$-szer folytonosan differenciálhatók, valamint létezik olyan $C \subset X$ kompakt halmaz, hogy minden*

$x \in X$-hez létezik olyan $y \in Y$, hogy $g_i(x) \in C$, továbbá teljesülnek a 2.1.1 (3) pontjának feltételei, akkor minden Lebesgue mérhető vagy Baire-tulajdonságú f megoldás egyben p-szer folytonosan differenciálható ($1 \leq p \leq \infty$).

2.1.5. Tétel. *A (2.1.1) jelöléseivel ha $t = 1, n = 2, g_1(x, y) \equiv y$ és a h függvény p-szer, a g_i függvények $\max\{2, p\}$-szer folytonosan differenciálhatók, akkor minden Lebesgue mérhető vagy Baire-tulajdonságú f megoldás egyben p-szer folytonosan differenciálható ($1 \leq p \leq \infty$). pedig*

2.1.6. Tétel. *Ha a (2.1.1) jelölései mellett az egyenlet*

$$f(x) = \sum_{i=1}^{n} h_i\left(x, y, f\left(g_i\left(x, y\right)\right)\right) \tag{2.2}$$

speciális formájú, ahol $h_i : D \times Z \to \mathbb{R}^t$ p-szer, a g_i függvények pedig $\max\{2, p\}$-szer folytonosan differenciálhatók, akkor akkor minden Lebesgue mérhető vagy Baire-tulajdonságú f megoldás egyben p-szer folytonosan differenciálható ($1 \leq p \leq \infty$).

2.1.7. Tétel. *Ha a (2.1.1) jelölései mellett $t = 1$, $D =]a, b[\times]a, b[$ és az egyenlet a*

$$f(x) = \sum_{i=1}^{n} c_i f\left(g_i\left(x, y\right)\right), \text{ amikor } (x, y) \in D \tag{2.3}$$

speciális alakú, ahol c_i rr, ($i = 1, 2, \ldots, n$),

$$g_i : D \to]a, b[$$

az ismert függvények továbbá a következő feltételek teljesülnek:

1. *$x, y \in]a, b[$ esetén $g_i(x, y)$ az x és y közé esik;*
2. *g_i analitikus és valamilyen $0 < A < 1$ konstanssal $x, y \in]a, b[$ esetén*

$$\left| \frac{\partial^p g_i}{\partial^k x \partial^{p-k} y}\left(x, y\right) \right| \leqq A^p p! \text{ és } p = 1, 2, \ldots ; , \ldots ;$$

3. *Minden $x \in]a, b[$ és $i = 1, 2, \ldots, n$ esetén az $y \to g_i(x, y)$ leképezés $]a, b[$-ról $]a, b[$-ra szigorúan monoton és ezen leképezés $\bar{g}_i$ inverze folytonosan differenciálható értelmezési tartományán.*

Ekkor minden Lebesgue mérhető vagy Baire-tulajdonságú f megoldás analitikus.

A fenti tételek bizonyítása megtalálható Járai [29]-ben.

2.2. Kompozit egyenletek regularitása

Egy függvényegyenletet kompozit egyenletnek nevezünk, ha az ismeretlen függvény vagy függvények előfordulnak ismeretlen függvény vagy függvények argumentumaként. Ezt a területet csak a teljesség kedvéért említjük, mivel nem tartozik szorosan témánkhoz. Az ilyen tipusú függvényegyenletekre, amennyiben az ismeretlen függvények monotonak, Járai [29] dolgozott ki egy metódust:

(I) a monotonitásból és magából a függvényegyenletből következik a belső függvények Jensen-konvexitása;

(II) a Bernstein-Doetsch tétel miatt a Jensen-konvexitásból következik a konvexitás, így a belső függvények egy megszámlálható halmaztól eltekintve differenciálhatók;

(III) A majdnem mindenhol differenciálható monoton függvényekre vonatkozó Lebesgue-tétel miatt a külső függvények deriválhatók, minek következtében a belső függvények folytonosan deriválhatók;

(IV) differenciálva az egyenletet, a kompozit rész kiküszöbölhető, a belső függvények deriváltjaira kapunk egy nem-kompozit egyenletet.

3. SZIMBOLIKUS INTERVALLUM ARITMETIKA

Mielőtt megvizsgáljuk az előző fejezetben szereplő regularitási tételek algoritmikus megközelítésének lehetőségeit, előtte egy olyan problémát tárgyalunk, ami egy igen hasznos segédeszköz lehet a függvényegyenletek átalakításában.

3.1. Szimbolikus intervallum aritmetika

A függvényegyenletek algebrai átalakításánál gyakran találkozzunk azzal a feladattal, hogy módosítanunk kell a függvények értelmezési tartományát és értékkészletét. Az általunk vizsgált esetekben az alapul szolgáló terek véges dimenziós euklideszi terek, a tartományok topológiailag „egyszerűek”, zárt, nyílt, kompakt halmazok. Egy dimenziós esetben ezek a tartományok többnyire intervallumok. A numerikus számításokban ma már klasszikusnak tekinthető módszer az intervallum analízis vagy intervallum aritmetika használata. Ez az alkalmazás azonban csak zárt intervallumokat használ és a numerikus pontosság a fontos. A mi esetünkben általában nyílt intervallumokkal dolgozunk és a numerikus pontosság nem túl lényeges. Ebben a részben tetszőleges intervallumok szimbolikus kezelésére alkalmas számítógépes megoldást ismertetünk.

3.1.1. A klasszikus intervallum analízisről röviden

3.1.1.1. Aritmetika

A klasszikus intervallum analízis a numerikus analízis hatékony módszere. 1966-ban Ramon E. Moore publikálta [33] -művét, ami máig sztenderd referenciája ennek a területnek. Azóta számtalan könyv született a témában, lásd [23, 24, 26, 27, 32, 34, 36]. Az itt szereplő tételek, állítások bizonyítása majd mindegyikben megtalálható. A klasszikus megközelítés lényegében alkalmas számunkra, nincsen szükségünk olyan általánosabb módszerekre, mint az irányított intervallumok, modális intervallumok vagy affin aritmetika. A klasszikus értelemben intervallumon a valós számegyenes egy összefüggő kompakt részhalmazát értjük. Mi a **végpont–jelölést** használjuk,

$X = [\underline{X}, \overline{X}]$, ahol $\underline{X} \leqq \overline{X}$ valós számok. Ha egyenlőség áll fönn, akkor **degenerált** intervallumról beszélünk. Ebben az esetben nyilvánvaló az intervallum azonosítása az $\underline{X} = \overline{X}$ valós számmal.

Először is szükségünk van az alapvető aritmetikai műveletekre. Ha X, Y intervallumok, akkor

$$X \odot Y = \{x \odot y : x \in X, y \in Y\}, \tag{3.1}$$

ahol $\odot \in \{-, +, \cdot, /\}$. Az osztáshoz pillanatnyilag feltételezzük, hogy $0 \notin Y$. A későbbiekben tárgyaljuk majd azt az esetet is, amikor az osztó tartalmazhatja a 0 számot. Ezen definíció alapján tudunk beszélni például egy intervallum reciprokáról:

$$1/Y = \{1/y : y \in Y\},$$

vagy definiálhatunk egyváltozós intervallumfüggvényeket is:

$$f(X) = \{f(x) : x \in X\},$$

ahol $f : \mathbb{R} \to \mathbb{R}$ egy függvény. Mivel az intervallumok a számegyenes részhalmazai, a halmazelméleti műveletek is értelmezhetjük az intervallumokon. Az természetesen nyilvánvaló, hogy két intervallum uniója nem lesz intervallum, ha metszetük üres. Ehelyett gyakran az intervallum–burkát használjuk inkább:

$$X \underline{\cup} Y = [\min(\underline{X}, \underline{Y}), \max(\overline{X}, \overline{Y})].$$

Az intervallumok összeadása és szorzása kommutatív, asszociatív, a $[0, 0]$ és $[1, 1]$ degenerált intervallum az additív illetve multiplikatív semleges elem. Inverzek általában nem léteznek, viszont az összeadásra érvényes az egyszerűsítési szabály. A disztributivitás helyett viszont csak az **szubdisztributivitási** tulajdonság teljesül, vagyis:

$$X(Y + Z) \subset XY + XZ, \tag{3.2}$$

ahol X, Y, Z intervallumok. Ez a tulajdonság kiemelt fontosságú, ez az alapja az intervallum analízis fundamentális tételének, amit később részletesebben tárgyalunk.

Most röviden összefoglaljuk, hogyan számíthatjuk ki a műveletek eredmény intervallumát. Jelölje $X = [\underline{X}, \overline{X}]$ és $Y = [\underline{Y}, \overline{Y}]$ azt a két intervallumot, amelyekkel a műve-

leteket végezzük. Könnyen ellenőrizhető, hogy

$$\begin{aligned} X+Y &= [\underline{X}+\underline{Y}, \overline{X}+\overline{Y}]\,, \\ -Y &= [-\overline{Y}, -\underline{Y}]\,, \\ X-Y &= X+(-Y) = [\underline{X}-\overline{Y}, \overline{X}-\underline{Y}]\,, \\ X\cdot Y &= [\min S, \max S],\ \text{ahol } S = \{\underline{X}\underline{Y}, \underline{X}\overline{Y}, \overline{X}\underline{Y}, \overline{X}\overline{Y}\}\,, \\ 1/Y &= [1/\overline{Y}, 1/\underline{Y}]\,, \text{if } 0 \notin Y\,. \end{aligned}$$

Az osztást a reciprokkal való szorzásként értelmezzük, ha $0 \notin Y$. Később ezt általánosítani fogjuk.
Szokásos még az intervallumok **középpont–sugár** reprezentációja. Ehhez szükségünk van néhány egyszerű fogalomra: az X intervallum $m(X)$ **középpontja** az $\frac{1}{2}(\underline{X}+\overline{X})$, $w(X)$ **szélessége** az $\overline{X}-\underline{X}$, $|X|$ **abszolút értéke** pedig az $\max(|\underline{X}|, |\overline{X}|$ valós szám. Ebben a reprezentációban a középpontot és a sugarat használjuk, ez utóbbi a szélesség fele.

3.1.1.2. Analízis

Mint azt az (3.1) egyenlet után említettük, ez a definíció alkalmas arra, hogy függvényeket definiáljunk intervallumokon. Természetes módon tudjuk a definíciót többváltozós esetre is általánosítani, nevezetesen, ha f egy függvény $\mathbb{R}^n$-en, valamint $X_1, \ldots, X_n$ $(n \in \mathbb{N})$ intervallumok, akkor

$$f(X_1, \ldots, X_n) = \{f(x_1, \ldots, x_n) : x_1 \in X_1, \ldots, x_n \in X_n\}\,.$$

Ez speciális esete a függvénykiterjesztésnek. Az így kapott intervallumfüggvényt gyakran **egyesített kiterjesztés**-nek hívjuk, mivel képe az eredeti függvény képpontjai egyesítése. Ezt a kiterjesztést általában nem triviális megtalálni, de néhány speciális esetben nem okoz problémát. Például, ha $f : \mathbb{R} \to \mathbb{R}$ monoton, úgy az X intervallum képe $[\min(f(\underline{X}), f(\overline{X})), \max(f(\underline{X}), f(\overline{X}))]$.
Azt gondolhatnánk, hogy ha definiáltunk egy függvényt valamilyen formulával, akkor a **természetes kiterjesztés**, azaz a valós változó intervallumváltozóra való formális

cseréjével készítjük az új függvényt, az egyesített kiterjesztést kapjuk. Ez azonban nem így van, sőt a helyzet még rosszabb, amiről az $f(x) = x(1-x)$ egyszerű valós függvény vizsgálatával nagyon könnyen meggyőződhetünk, lásd Moore et al. [35]. A kifejezés kiértékelési módjától függően különböző eredményez juthatunk. Helyettesítsük be az $X = [0,1]$ intervallumváltozót az x helyett! Ha az eredeti formulát értékeljük ki, az $X \cdot (1-X)$ művelet eredménye a $[0,1]$ intervallum lesz. Ha viszont először elvégezzük a beszorzást és a $X - X^2$ kifejezést értékeljük ki, akkor a $[-1,1]$ intervallumot kapjuk. Tehát az eredményül kapott intervallumfüggvények – ha a valós változót intervallumra cseréljük – különböznek. Itt jegyezzük meg, hogy még az intervallum négyzetre emelése is különbözik az önmagával vett szorzattól.
Azt mondjuk, hogy az F függvény az f valós függvény egy **intervallum kiterjesztése**, ha minden x esetén $F([x,x]) = f(x)$, vagy általánosabban több változóra, ha

$$F([x_1,x_1],\ldots,[x_n,x_n]) = f(x_1,\ldots,x_n)\,.$$

Az előzőek alapján azt mondhatjuk, hogy egy valós függvénynek nincs egyértelmű intervallum kiterjesztése. Nagyon könnyen látható, hogy $Y_i \subset X_i, i = 1,2$ esetén $Y_1 \odot Y_2 \subset X_1 \odot X_2$. Ez motiválja a következő definíciót:

3.1.1. Definíció. Az F intervallum függvényt **tartalmazás invariáns**-nak nevezzük, ha $Y_i \subset X_i, i = 1,\ldots,n, n \in \mathbb{N}$ esetén $F(Y_1,\ldots,Y_n) \subset F(X_1,\ldots,X_n)$.

Egy fontos függvényosztály kielégíti a fenti definíció feltételeit, a **racionális intervallumfüggvények**, ezek azok a függvények melyek véges sok intervallum aritmetikai művelet segítségével kiszámíthatók.
A következő tétel rávilágít az egyesített kiterjesztés központi szerepére az intervallum aritmetikában.

3.1.1. Tétel (Az intervallum analízis alaptétele)**.** *Ha F az f függvény tartalmazás invariánskiterjesztése, akkor*

$$f(X_1,\ldots,X_n) \subset F(X_1,\ldots,X_n)\,.$$

A tétel bizonyítását lásd például Moore et al. [35]. Ezen tétel alapján az egyesített kiterjesztésre úgy gondolhatunk, mint a „legkisebb” kiterjesztésre.

Intervallumok távolságát a következőképp tudjuk definiálni:

$$d(X,Y) = \max(|\underline{X} - \underline{Y}|, |\overline{X} - \overline{Y}|)\,. \tag{3.3}$$

Ez valóban metrika, a bizonyítás egyszerű számolás. Néha ezt a metrikát **Moore–metrikának** is nevezzük. nyilvánvaló, hogy ez a szokásos Hausdorff–távolság intervallumokra. A kapott metrikus tér ezért teljes, a valós számoknak a degenerált intervallumokkal való azonosítása pedig egy izometrikus beágyazás. A definíció alapján könnyen bizonyítható az is, hogy egy X_k intervallumsorozat pontosan akkor konvergál egy X intervallumhoz, ha az X_k-k végpontjai az X végpontjaihoz konvergálnak.

3.1.2. Definíció. Azt mondjuk, hogy egy F intervallumfüggvény **Lipschitz** tulajdonságú az X intervallumon, ha létezik egy L nemnegatív valós szám, melyre

$$w(F(x)) \leqq Lw(X)\,, \tag{3.4}$$

aho w az intervallum szélessége.

Könnyen láthatók az alábbi

Tények.

1 Valós racionális függvény természetes kitejesztése Lipschitz.

2 Ha az f valós függvény Lipschitz az X_0 intervallumon, akkor a egyesített kiterjesztés Lipschitz az X_0-on-

3.1.2. Implementációk

Nagyszámú számítógépes megvalósítása létezik az intervallum aritmetikának, ezek mindegyike numerikus célú. A Interval and Related Software honlapon (lásd [18]) egy rendkívül átfogó listát találhatunk ezekről az implementációkról, számos könyv és cikk is foglalkozik velük, így például az Moore et al. [35] példái a MatLab IntPak csomagjában lettek leprogramozva. Mivel bennünket a szimbolikus számítások érdekelnek, szólunk néhány szót a MAPLE és a SAGE implemetációiról.

Intervallumok a MAPLE -ben

A Maple hivatalos kiadásaiban található megoldás nem igazán kielégítő, ráadásul egyszerre több megközelítést is alkalmaztak.

Az INTERVAL objektum: Találhatunk egy *INTERVAL()* objektumot, ami lényegében dokumentálatlan sajátossága a MAPLE -nek. A rendszer hagyományaihoz híven természetesen ez egy polimorf objektum. Segítségével intervallumok sorozatát definiálhatjuk a *INTERVAL*$(a..b, c..d, \ldots)$ szintaxis segítségével, vagy pedig intervallum-változót készíthetünk a *INTERVAL*$(x, a..b)$ formában. Itt a határok mindegyike lehet végtelen is. Az első alak alkalmas arra, hogy az *evalr()* függvény paramétere illetve visszatérési értéke legyen. Egy példával illusztrálva: a $evalr\left(sin\left(\textit{INTERVAL}\left(2..7\right)\right)\right)$, kifejezés értéke $[-1..\sin\left(2\right)]$. Másik példaként értékeljük ki MAPLE -ben az $evalr(|x|)$ kifejezést. Az eredmény

$$\textit{INTERVAL}\left(\textit{INTERVAL}\left(x, 0..\infty\right), -\textit{INTERVAL}\left(x, -\infty..0\right)\right) .$$

Sajnos, a második szintaxisra nem találtunk működő példát. Ehhez az objektumhoz tartozik még a *shake()* függvény, ami egy adott értékhez megadott pontosságú befoglaló intervallumot generál. Így például a $shake(e, 5)$ értéke *INTERVAL*$(2.718010..2.718554)$ lesz. Fontos megjegyezni, hogy ezekkel az objektumokkal tudunk aritmetikai műveleteket végezni.

A range objektum: Az előző részben is megtalálható a MAPLE egy „rejtett” konstrukciója, a „..” típus, amire a MAPLE „range” objektumként hivatkozik. Szintaxisa: $a..b$, ahol a, b tetszőleges valós számok. Ezt akár egy intervallum-konstansnak is tekinthetnénk, alkalmazása azonban erősen korlátozott. Alapvetően két bevált alkalmazása van. Ha a végpontok egészek, akkor a *seq(), sum()* jellegű függvényekben mint iterátort használjuk. A másik felhasználása pedig a *plot()* típusú grafikus függvényben a megjelenítés intervallumának megadása.

A RealRange() függvény: Első pillanatban ez tűnik céljainkhoz legközelebb állónak, mivel ezzel nem csak zárt intervallumok adhatók meg, ugyanis a végpont megadásánál használhatunk egy *Open()* függvényt is. Ez a MAPLE „assume” rendszerének belső függvénye, lényegében teljesen dokumentálatlan. Ehhez a

konstrukcióhoz nem léteznek operátorok, a belső automatikus egyszerűsítések pedig néha meghökkentő eredményre vezetnek.

az intpakX csomag: Ez egy a Wuppertali Egyetemen készített igen kiváló csomag, azonban csak zárt intervallumokkal dolgozik és kifejezetten numerikus célokra használható. Bővebben lásd [30].

Intervallumok a SAGE -ben

A SAGE numerikus számításokhoz támogatja a tetszőleges pontosságú valós- és komplex intervallum analízist. Ez az MPFI szabad szoftver implementálása. Meglepő módon a középpont–sugár reprezentációt használja; ez a SAGE -terminológia szerinti „kérdőjeles" ábrázolás, a $1.414213562373095?$ jelölés azt jelenti, hogy az utolsó számjegyben valószínűleg hiba van. Bővebben [2, 3].

3.1.3. Tetszőleges intervallumok kezelése

Áttekintve az előzőeket, a zárt intervallumokra történő korlátozás többnyire nem szükséges. Ha áttérünk tetszőleges intervallumokra, a következő kritikus pontokkal találkozunk:

Problémák.

1 *A degenerált nyílt illetve félig zárt intervallumok azonosítása a valós számokkal nem lehetséges.*

2 *A Moore-távolság csak kvázimetrika lesz, így tetszőlges intervallumokból ezzel nem tudunk teljes metrikus teret készíteni.*

3 *A végpontok konvergenciája problematikus.*

4 *A Lipschitz-feltétel öröklődése nem garantált.*

Ezek a problémák nem veszélyesek célkitűzéseinkre. Fő célunk a racionális függvények kezelése, ebben az esetben a (2)-(4) problémák nem jelentősek. A legérdekesebb az első probléma. Mivel ebben az esetben üres halmazt kapunk, ezt nem tudjuk azonosítani valós számmal. A legkényelmesebb megoldás az, ha az intervallumokat a továbbiakban rendezett számpároknak tekintjük.

A műveletek

Ha visszatekintünk 3.1 definícióra, láthatjuk, hogy ez nem függ az intervallum típusától, ezek algebrai műveletek a valós számegyenes részhalmazaival. Hasonló halmazműveletek jól ismertek a matematika más területein is, így például a konvex geometriában vagy a csoportelméletben, a gyűrűelméletben is. Intervallumok esetében a műveleteket egyszerűen elvégezhetjük a végpontok segítségével. Ha csak tisztán nyílt vagy tisztán zárt intervallumokkal dolgozunk nincs szükségünk semmilyen egyéb megfontolásra, de nem túl nehéz kombinálni a különböző intervallum típusokat. Egyetlen megoldandó feladatunk van, el kell döntenünk, az eredmény intervallum végpontjai milyenek lesznek. A nyílt végpontok „mohók", lényegében „elnyelik" a zárt intervallumokat. Erről a kérdésről precízebben és részletesebben fogunk a későbbiekben beszélni.
A továbbiakban azt fogjuk mondani, hogy egy intervallum végpont nyílt vagy zárt, aszerint, hogy hozzátartozik-e az intervallumhoz vagy nem. A következőkben részletesebben foglalkozunk az intervallum aritmetikai műveletekkel. Ha $x \in \mathbb{R}$ egy intervallum végpontja, akkor $Open(x) = true$, ha x nyílt, $Open(x) = false$, ha x zárt végpont. A továbbiakban leírt műveletek kezelik a nemkorlátos intervallumokat is.

Összeadás Mint az előzőekben láttuk, az X, Y intervallumok összeadásakor az eredmény végpontjai $\underline{X} + \underline{Y}$ illetve $\overline{X} + \overline{Y}$. Természetesen, ha valamelyik végpont nyílt, az eredmény végpont is az. Precízebben: ha $Z = X + Y$, akkor

$$Open(\underline{Z}) = Open(\underline{X}) \vee Open(\underline{Y})$$
$$Open(\overline{Z}) = Open(\overline{X}) \vee Open(\overline{Y}),$$

ahol $\vee$ a logikai vagy művelet.
A nem korlátos intervallumok esete a 3.1 táblázatban található a 37 lapon. Itt a, b tetszőleges bővített valós számok.

+	$-\infty$	a	∞
$-\infty$	$-\infty$	$-\infty$	undef.
b	$-\infty$	$a+b$	∞
∞	undef.	∞	∞

3.1. táblázat: Összeadás végtelen értékekkel

Kivonás Ebben az esetben a műveletet az ellentétes végpontokkal kell végrahajtanunk, így ha $Z = X - Y$, akkor $X - Y = X + (-Y) = [\underline{X} - \overline{Y}, \overline{X} - \underline{Y}]$ és

$$Open(\underline{Z}) = Open(\underline{X}) \vee Open(\overline{Y})$$
$$Open(\overline{Z}) = Open(\overline{X}) \vee Open(\underline{Y})\,.$$

A végtelen értékekkel való számolást a 3.2 táblázatban találjuk a 37 lapon.

-	-∞	a	∞
$-\infty$	undef.	$-\infty$	$-\infty$
b	∞	$b-a$	$-\infty$
∞	∞	∞	undef.

3.2. táblázat: Kivonás végtelen értékekkel

Szorzás A $Z = X \cdot Y = [\min S, \max S]$, definíció alapján, ahol

$$S = \{\underline{X}\,\underline{Y}, \underline{X}\,\overline{Y}, \overline{X}\,\underline{Y}, \overline{X}\,\overline{Y}\}\,,$$

a szorzás viszonylag egyszerűen végrehajtható. Bár az általunk tárgyalt komputeres alkalmazások nem túl időigényesek, mégis illendő a szorzás esetét megvizsgálnunk. Jól ismert tény, lásd Kulisch and Karlsruhe [32], Moore et al. [35], hogy nincs négy szorzásra szükségünk, a legrosszabb esetben is elegendő három. Abban az esetben, ha a nullát legalább az egyik intervallum nem tartalmazza, elegendő két szorzás, amint a 3.3 táblázatból láthatjuk.

Ebben az esetben az *Open()* függvény értékeét könnyű számolni, mivel értéke az aktuális számolásban használt végpontok *Open()* értékének logikai vagy-ja.

Case	$\underline{Z}$	$\overline{Z}$
$0 \leqq \underline{X}$ and $0 \leqq \overline{Y}$	$\underline{X} \cdot \underline{Y}$	$\overline{X} \cdot \overline{Y}$
$\underline{X} < 0 < \overline{X}$ and $0 \leqq \underline{Y}$	$\underline{X} \cdot \overline{Y}$	$\overline{X} \cdot \overline{Y}$
$\overline{X} <\leqq 0$ and $0 \leqq \underline{Y}$	$\underline{X} \cdot \overline{Y}$	$\overline{X} \cdot \underline{Y}$
$0 \leqq \underline{X}$ and $\underline{Y} < 0 < \overline{Y}$	$\overline{X} \cdot \underline{Y}$	$\overline{X} \cdot \overline{Y}$
$\overline{X} < 0$ and $\underline{Y} < 0 < \overline{Y}$	$\underline{X} \cdot \overline{Y}$	$\underline{X} \cdot \underline{Y}$
$0 \leqq \underline{X}$ and $\overline{Y} \leqq 0$	$\overline{X} \cdot \underline{Y}$	$\underline{X} \cdot \overline{Y}$
$\underline{X} < 0 < \overline{X}$ and $\overline{Y} \leqq 0$	$\overline{X} \cdot \underline{Y}$	$\underline{X} \cdot \underline{Y}$
$\underline{X} < 0 < \overline{X}$ and $\overline{Y} \leqq 0$	$\overline{X} \cdot \underline{Y}$	$\underline{X} \cdot \underline{Y}$
$\underline{X} \leqq 0$ and $\underline{Y} \leqq 0$	$\overline{X} \cdot \overline{Y}$	$\underline{X} \cdot \underline{Y}$
$\underline{X} < 0 < \overline{X}$ and $\underline{Y} < 0 < \overline{Y}$	See later	

3.3. táblázat: Intervallumszorzás 1

Abban az esetben, ha mindkét intervallum tartalmazza a 0-t, valóban szükségünk van három szorzásra. A számítási folyamat a 3.4 táblázatban található. Az *Open()* értékek

Case	$\underline{Z}$	$\overline{Z}$
$0 \leqq \lvert\underline{X}\rvert \leqq \overline{X}$ and $0 \leqq \lvert\underline{Y}\rvert \leqq \overline{Y}$	$\min\{\underline{X} \cdot \overline{Y}, \overline{X} \cdot \underline{Y}\}$	$\overline{X} \cdot \overline{Y}$
$0 \leqq \overline{X} \leqq \lvert\underline{X}\rvert$ and $0 \leqq \overline{Y} \leqq \lvert\underline{Y}\rvert$	$\min\{\underline{X} \cdot \overline{Y}, \overline{X} \cdot \underline{Y}\}$	$\underline{X} \cdot \underline{Y}$
$0 \leqq \lvert\underline{X}\rvert \leqq \overline{X}$ and $0 \leqq \overline{Y} \leqq \lvert\underline{Y}\rvert$	$\overline{X} \cdot \underline{Y}$	$\max\{\underline{X} \cdot \underline{Y}, \overline{X} \cdot \overline{Y}\}$
$0 \leqq \overline{X} \leqq \lvert\underline{X}\rvert$ and $0 \leqq \overline{Y} \leqq \lvert\underline{Y}\rvert$	$\underline{X} \cdot \overline{Y}$	$\max\{\underline{X} \cdot \underline{Y}, \overline{X} \cdot \overline{Y}\}$

3.4. táblázat: Intervallumszorzás 2

számítása magától értetődő, mindazonáltal jobban áttekinthető programkód formájában, mint matematikai kifejezésként, így arrra realizálásnál visszatérünk. Hátra van még a végtelen értékek kezelése, ez a 3.5 táblázatban látható.

Megjegyezzük, hogy itt eltérünk az IEEE 754 standardtól. A standard javaslata szerint a zéró és a végtelen értékek szorzata definiálatlan (not a numnber, NaN), számunkra azonban célszerűbb ha zérusnak tekintjük.

$\cdot$	$-\infty$	$b<0$	0	$b>0$	∞
$-\infty$	∞	∞	0	$-\infty$	$-\infty$
$a<0$	∞	classical			$-\infty$
0	0	computing with			0
$a>0$	$-\infty$	finite values			∞
∞	$-\infty$	$-\infty$	0	∞	∞

3.5. táblázat: Multiplying infinite values

Osztás Ha az 3.1 definíciót alkalmazzuk az intervallumok osztására, akkor a következő kifejezést kapjuk:

$$X/Y = \{x/y : x \in X \wedge y \in Y\},$$

hacsak $0 \notin Y$. Mint korábban említettük, ebben az esetben a művelet nagyon egyszerű, csak az Y intervallum reciprokával kell szoroznunk. Átalakíthatjuk az előző kifejezést úgy, hogy ne tartalmazzon osztást:

$$X/Y = \{z : z \cdot y = x \wedge x \in X \wedge y \in Y\}. \tag{3.5}$$

Nyilvánvaló, ha $0 \notin Y$, akkor ez ekvivalens az első megfogalmazással. Két triviális esetet gyorsan át tudunk tekinteni. Először is, ha $0 \in X$, akkor a fenti 3.5 átfogalmazásból következik, hogy az eredmény az egész számegyenes (mert x, $x \cdot 0$ is 0). A másik trivialitás az, amikor $0 \notin X$ és $Y = [0, 0]$. Ekkor nincs olyan nemzérus a valós szám, amivel $0 \cdot x = a$, így az eredmény az üres halmaz. A többi eset hasonló a szorzás eseteihez, azt kell megvizsgálnunk, a két intervallum hogyan helyezkedik el a nullához képest. A részletek a 3.6 táblázatban találhatók. Itt a kúpos zárójelek tetszőleges típusú intervallum végpontot jelölnek. Tisztáznunk kell azokat az eseteket, amikor a táblázat első oszlopában végtelen értékek szerepelnek. A szabály egyszerű: ha nullát vagy végtelen értéket végtelen értékkel osztunk, akkor az eredmény definiálatlan, különben nulla.

3.6. táblázat: Intervallum osztás

Case	Result set
$\overline{X} < 0$ and $\underline{Y} < \overline{Y} = 0$	$\langle \overline{X}/\underline{Y}, \infty)$
$\overline{X} < 0$ and $\underline{Y} < 0 < \overline{Y}$	$(-\infty, \overline{X}/\overline{Y}\rangle \cup \langle \overline{X}/\underline{Y}, \infty)$
$\overline{X} < 0$ and $0 = \underline{Y} < \overline{Y}$	$\langle -\infty, \overline{X}/\overline{Y}\rangle$
$\underline{X} > 0$ and $\underline{Y} < \overline{Y} = 0$	$(-\infty, \underline{X}/\underline{Y}\rangle$
$\underline{X} > 0$ and $\underline{Y} < 0 < \overline{Y}$	$(-\infty, \underline{X}/\underline{Y}\rangle \cup \langle \underline{X}/\overline{Y}, \infty)$
$\underline{X} > 0$ and $0 = \underline{Y} < \overline{Y}$	$\langle \underline{X}/\overline{Y}, \infty)$

A topológiáról

Megjegyezzük, hogy a klasszikus intervallum analízis említett fogalmai függetlenek az intervallum típusától. Ez azt jelenti, hogy az olyan fogalmakat, mint egyesített kiterjesztés, természetes kiterjesztés, intervallum kiterjesztés, tartalmazás invariáns, racionális függvény, változtatás nélkül használhatjuk. A fundamentális tétel is igaz marad, mivel bizonyítása csakis egyszerűbb halmazelméleti megfontolásokat tartalmaz. A Moore–távolság a továbbiakban is használható, bár csak kvázimetrikus térben gondolkodhatunk. A Lipschitz feltétel az érdekesebb számunkra, de ez racionális függvényekre nyilvánvaló.

3.1.4. Realizáció

A Czirbusz [20]-ben készült egy egyszerűbb implementáció a MAPLE *RealRange()* konstrukciója segítségével. Azonban, mint az előzőekben említettük, ez egy szinte dokumentálatlan tulajdonság. Másrészt teljesen MAPLE -specifikus, ezért ezt ebben a megvalósításban elvetettük. Ehelyett olyan megoldást választottunk, amely más környezetekben is könnyen implementálható. Így a MAPLE modul-technikáját választottuk, ami a modern nyelvek objektum orientált technológiájára hasonlít. Megjegyezzük, hogy a MAPLE legújabb verziói már tartalmaznak objektumokat, ez azonban nem áll rendelkezésünkre. Az intervallumok struktúráját a (Az intervallum objektum) programlistában definiáltuk.

Itt definiáltuk a korábban említett aritmetikai függvényeket is. A *ModulePrint* és *ModuleApply* függvények segítségével valósítható meg a MAPLE -ben megszokott ekhózás: az intervallumot a szokásos jelölésmódban írja vissza az output sorba. A *isInterval* változó segítségével jelezzük más procedúrák felé, hogy intervallum objektumról van szó.
Magát az intervallum típust a következőképp definiáljuk:

Az intervallum típus

```
isInterval := proc (x::anything)
  try
    x:-isInterval
  catch:
    false
  end try
end proc

TypeTools:-AddType(Interval, IsInterval)
```

Az intervallumok szorzása a (Intervallumszorzás) programlistában található.
A szokásos szorzást az *override* konstrukcióval felülírtuk, ezért úgy szorozhatunk intervallumokat, mint közönséges számokat. A kód megvalósítja a vegyes műveletet is, vagyis intervallum és szám szorzatát is. (Ugyanez a függelékben SAGE -nyelven is megtalálható: (**??**))

3.1.5. Alkalmazások

Említettük már az előzőekben, hogy az intervallum kiterjesztés nem egyértelmű. Az $X \cdot (1 - X)$ kifejezést kétféleképpen számíthatjuk. MAPLE -ben:

The Interval Function $X \cdot (1 - X)$

```
    with(IntervalOperators);
        [`*`, `+`, `-`, `/`, `<`, `>`, Hull,
```

```
     Intersect, Reciprocial, Union, `^`]
```

$X := Interval(0, 1, false, false);$

```
    X := "[0,1]"
```

$X \cdot (1 - X);$

```
    "[0,1]"
```

$X - X^2;$

```
    "[-1,1]"
unwith(IntervalOperators);
```

Példa az alapműveletekre:

Alapműveletek

$A := Interval(1, 2, false, false); B := Interval(5, 4);$

```
    A := "[1,2]"
    B := "(4,5)"
```

$A \cdot B;$

```
    "(4,10)"
```

$\frac{A}{2};$

```
    "[1/2,1]"
```

$\frac{1}{B};$

```
    "(1/5,1/2)"
```

$\frac{A}{B};$

```
    "(1/5,1/2)"
```

4. A REGULARITÁS VIZSGÁLATA

A következőkben tehát az

$$f(x) = h\Big(x, y, f\big(g_1(x,y)\big), \ldots, f\big(g_n(x,y)\big)\Big) \tag{4.1}$$

speciális alakú függvényegyenleteket vizsgáljuk. Ez a típusú egyenlet mindazonáltal eléggé általános, sok ismert függvényegyenlet ezt a sémát követi. Megjegyezzük, hogy az esetek többségében a h külső függvény nem igazán bonyolult, többnyire valamilyen aritmetikai kifejezés, így simasága a priori adott. Sokszor igaz ez a g_i belső függvényekre is. Emiatt a legfontosabb feladat a parciális deriváltak mátrixai rangjának megállapítása. Először a 2.1.6 tétel feltételeit kielégítő egyenletekkel foglalkozunk. Ezután a 2.1.1 parciális deriváltakra kirótt feltétel geometriai interpretációjával foglalkozunk, majd egy nem kvázi-lineáris egyenletnél megkeressük a regularitáshoz szükséges kompakt tartományt.

4.1. Kvázi-lineáris egyenletek

Az

$$f(x) = \sum_{i=1}^{n} h_i\left(x, y, f\left(g_i(x,y)\right)\right) \tag{4.2}$$

függvényegyenlet regularitási problémáját a 2.1.6 tétel alapján tudjuk vizsgálni. Az ottani általános megfogalmazáshoz képest egy egyszerűbb problémát vizsgálunk, feltételezzük, hogy $r = s = t = 1$. Ez erős korlátozásnak tűnik, de sok érdekes egyenlet tartozik ebbe a csoportba. Az ebben a szakaszban bemutatottak Czirbusz [19]-ben jelentek meg cikk formájában

A számítógépes megvalósításhoz a MAPLE komputeralgebra rendszer 15-ös verzióját használtuk, de a kód lefut a legalább tízes fölötti verziójú rendszerek mindegyikén, a procedúra paraméterezésben innentől van lehetőségünk a pozícionális paraméterek használatára, programozói kényelemből van rá szükségünk. A program teljes forráskódja letölthető a dolgozat írójának honlapjáról.

A megvalósított program inputja a 2.1.1 probléma szerinti vizsgálandó függvényegyenlet, illetve opcionálisan a D értelmezési tartomány, amit egyenlőtlenségek formájában adhatunk meg. Az output a „rossz helyek" listája, azaz az értelmezési tartomány azon részhalmaza, ahol a rang-kritérium nem teljesül. A használt jelöléseket (belső, külső függvény, változók nevei) globális változókkal tehetjük meg, amennyiben a 2.1.1 jelöléseitől eltérünk. A következőkben a program lépéseit ismertetjük.

4.1.1. A főprogram

A főprogram kódja (A rangkritérium ellenőrzésének főprogramja) programlistában található. A főprogram egy inicializáló részben a **Newtask()** procedúra meghívásával végzi el a paraméterbeállításokat, és definiálja a később használandó **CNT** táblát, amit majd az **Elimf()** procedúra tölt fel az intervallumokká konvertált tartomány definíciókkal. Ezután egy ciklusban először az **Elimf()** függvényt hívja meg, majd egy „parser"-t és ellenőrzi, hogy ennek eredménye függ-e az y változótól. Az **Elimf()** procedúra lényegében egy egyszerű pszeudo-megoldó, amit röviden a következőkben ismertetünk.

4.1.2. A pszeudo-megoldó

Az (A pszeudo-megoldó kódja) eljárás megpróbálja az egyenletet (4.1) formára alakítani. Nem közvetlen eliminációt alkalmaz, hanem a heurisztikus függvényegyenlet megoldásokhoz hasonló egyszerű helyettesítéseket hajt végre. Jelen verzió az $x \pm y, x \cdot y, x/y,$ ls y/x alakú kifejezéseket kezeli. Az eredetileg egyenlőtlenségek segítségével megadott értelmezési tartományt újraszámolja, ezt a 3 fejezetben ismertetett szimbolikus intervallumkezeléssel végzi.

4.1.3. A parser

A parser feladata a függvényből egy listába összegyűjteni a belső függvényeket. A már (4.1) formájúvá alakított egyenletet sztringgé konvertálja és „kiszedi" a g_i függvényeket a zárójelekből. A szöveggé konvertálás oka az, hogy a függvények kezelése minden komputeralgebra rendszer „belügyének" számít, a szövegek feldolgozása viszont minden rendszeren megvalósítható. A kód itt (A parser kódja) található.

4.1.4. A rangkritérium ellenőrzése

A harmadik lépésben a g_i-k deriváltjaiból felépítjük a Jacobi-mátrixokat és kiszámítjuk a rangjukat. A szimbolikus intervallumkezelést leszámítva ez az a rész, ahol ténylegesen szükségünk van a komputeralgebrai rendszerek matematikai képességeire, használjuk a MAPLE **diff** és **solve** utasításait. A kód itt (A rang ellenőrzése) található.

Néhány segédprogramot is használunk, például azon egyenlőtlenségek megoldására, melyeket a beépített **solve** nem kezel.

4.1.5. A vizsgált függvényegyenletek

A vizsgált függvényegyenletek többnyire a Aczél [5], Járai [29] könyvekből származnak. A D értelmezési tartomány , ha nincs feltüntetve, akkor az $\mathbb{R}^2$ valós számsík.
Az additív Cauchy egyenlet $f(x+y) = f(x) + f(y)$. A program először $f(x) = f(x+y) - f(y)$alakra konvertálja, nyilvánvaló a rangkritérium teljesülése.
A Cauchy-féle hatványegyenlet $f(xy) = f(x) f(y)$. A segédprocedúrák hívása csak az illusztráció kedvéért történik:

```
NewTask();
>FE := f(x*y) = f(x)*f(y);
                f(x y) = f(x) f(y)
>Elimf(FE)[1][1];
           f(x) = f(x/y)*f(y)
>FECheck(FE);
                {{x = 0, y = y}}
```

Először töröljük a változókat, majd átalakítjuk a megfelelő formátumra az egyenletet, aztán ellenőrizzük a rangot; a válasz szerint $x = 0$ esetén minden y rossz.
Néhány nevezetes függvényegyenletre a rang mindenhol maximális:
A Jensen egyenlet $f((x+y)/2) = (f(x) + f(y))/2$.
Általánosított Cauchy egyenlet $f(x+y) = h(x, y, f(x), f(y))$.
Az információelmélet alapegyenletet (Lásd: Aczél and Daróczy [6])

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

Ez utóbbinál a $D = \{(x, y) : 0 < x, y < 1, x + y < 1\}$ tartományon teljesül a rangkritérium. A program jelenleg az információelmélet általánosított egyenletét csak abban az esetben kezeli, ha az ismeretlen függvények azonosak.

A dilogarithm egyenlet $f(x) + f(y) + f(1 - xy) + f\left(\frac{1-x}{1-xy}\right) + f\left(\frac{1-y}{1-xy}\right) = 0$. A $D = \{(x, y) : 0 < x, y < 1\}$ tartományon teljesül a rangkritérium.

A (2,2) additivitási egyenlet

$$f(xy)+f\left(\frac{x}{1-y}\right)+f\left(\frac{1-y}{x}\right)+f\left(\frac{1-x}{1-y}\right) = f(x)+f(1-x)+f(y)+f(1-y)$$

Ebből az egyenletből $f(x)$ közvetlenül kifejezhető, de az új egyenletben nem minden belső függvény függ az y változótól. Lépésről lépésre haladva:

```
> NewTask();
> FE := f(x*y)+f(x/(1-y))+f((1-y)/x)+f((1-x)/(1-y)) =
          f(x)+f(1-x)+f(y)+f(1-y):
> DD := [0 < x and x < 1, 0 < y and y < 1];
                [0 < x and x < 1, 0 < y and y < 1]
>Elimf(FE, DD, 0)[1][1]:
>Parse(rhs(%))
  [[x*y], [x/(1-y)], [(1-y)/x], [(1-x)/(1-y)], [1-x], [y], [1-y]]
>Elimf(FE, DD, 1)[1][1]:
>Parse(rhs(%))
  [[(1-x)*y], [(1-x)/(1-y)], [(1-y)/(1-x)],
   [x/(1-y)], [1-x], [y], [1-y]]
>Elimf(FE, DD, 2)[1][1]:
>Parse(rhs(%))
  [[(1-x+x*y)*y], [(1-x+x*y)/(1-y)], [(1-y)/(1-x+x*y)],
   [1-y], [1-x+x*y], [x-x*y], [y]]
```

A program megtalálja a célravezető $t = xy$ helyettesítést. Elhagyva a hosszú részszámításokat, a deriváltakra a következőt kapjuk:

```
DG := {[-1], [1], [x/y^2], [-x/y^2], [-y/x+(1-y)/x],
       [-x/(y^2*(1-y))+x/(y*(1-y)^2)],
       [x/(y^2*(1-y))+(1-x/y)/(1-y)^2]}
```

Megjegyezzük, hogy a MAPLE szimbolikus képességei miatt az $f(x)+\sum_{i=1}^{n} c_i f(A_i x + B_i y) = 0$ alakú egyenletek is vizsgálgatók, ahol $A_i,\ B_i$ és c_i az x, y változóktól független szimbólumok.

4.2. Geometriai interpretáció

A 2.1.1 probléma 3. pontjában a parciális deriváltak mátrixainak rangjára megfogalmazott feltétel geometriai reprezentációjával fogunk ebben részben. Tekintsük először az $r = s = t = 1$ esetet. Ebben az esetben a rangkritérium arra a feltételre egyszerűsödik, hogy $\forall x \in X$ esetén van olyan $y \in Y$, melyre

$$\frac{\partial g_i(x,y)}{\partial y} \neq 0\,.$$

A parciális deriváltat egy $x \to \frac{\partial g_i(x,y)}{\partial y}$ leképezésnek tekintve, akkor ez egy görbe az $\mathbb{R}^2$ síkon, a feltételünk pedig azt jelenti, hogy a görbe nem tartalmazhat semmilyen az x-tengelyre merőleges egyenes szakaszt. Megfordítva, ha a görbe egyetlen x-re sem tartalmaz függőleges szakaszt, akkor az X-en vett metszéspontok összessége egy seholsem sűrű részhalmaza $D \subset \mathbb{R}^2$-nek, így a regularitási feltétel teljesül. A következőkben először ezt illusztráljuk néhány példával.

4.2.1. Az $r = s = t = 1$ eset

A belső függvények többnyire igen egyszerűk, általában az x és y változó első- vagy másod fokú polinomjai, vagy ezen polinomok hányadosai. Ekkor nincs szükségünk a mátrix rangjának kiszámítására, a feltétel teljesülése „látszik”.

4.2.1. Példa. Az $f(x + y) = h(x, y, f(x), f(y))$ alakú egyenletben, ahol f az ismeretlen, h pedig az ismert függvény, a belső függvények $[y, x - y]$, melyet y szerint deriválva $[1, -1]$-et kapunk, látható, hogy a ez semmilyen y értékre sem lehet zérus.

4.2.2. Példa. Az $f(x)+(1-x)f(\frac{y}{1-x}) = f(y)+(1-y)f(\frac{x}{1-y})$ függvényegyenlet belső függvénye: $[y, \frac{x}{1-y}, \frac{y}{1-x}]$. Deriválással a $[1, \frac{x}{(1-y)^2}, \frac{1}{1-x}]$ függvényeket kapjuk, amiből azonnal látszik a regularitási feltétel teljesülése.

4.2.3. Példa. A függvényegyenlet belső függvényei $[y, xy, \frac{x}{1-y}, \frac{1-x}{1-y}, \frac{1-y}{x}, 1-x, 1-y]$, a deriváltak pedig $[1, x, \frac{x}{(1-y)^2}, \frac{1-x}{(1-y)^2}, -x^{-1}, 0, -1]$, ahol a hatodik elem azonosan zérus, tehát a regularitás ezen függvények segítségével nem igazolható.

4.2.2. Általánosabb függvényegyenletek

4.2.4. Példa. Tekintsük az

$$f(x) = \frac{f(y)\left(f\left(t\left(x+y\right)\right)-f\left(tx\right)\right)}{f\left(t'\,x'\right)-f\left(t'\,y'\right)} + \frac{\left(f\left(tx\right)-f\left(ty\right)\right)\left(f\left(x'\right)\left(f\left(t'\ \left(x+y\right)\right)-f\left(t'\,y'\right)\right)-f\left(y'\right)\left(f\left(t'\ \left(x+y\right)\right)-f\left(t'\,x'\right)\right)\right)}{\left(f\left(t'\,x'\right)-f\left(t'\,y'\right)\right)\left(f\left(t\left(x+y\right)\right)-f\left(ty\right)\right)}$$

függvényegyenletet. Az egyszerűség kedvéért MAPLE -ben az x', y', t' változókat rendre $x1, y1, t1$ jelöli. Ezek segítségével a belső függvények

$$\left[y, y1, tx, ty, t\left(x+y\right), t1\ y1, t1\ \left(x+y\right), t1\ \left(x+y-y1\right), x+y-y1\right],$$

míg a deriváltakból alkotott Jacobi-mátrixok sorozata:

$$[1,0,0,0],[0,0,1,0],[0,x,0,0],[t,y,0,0],[t,x+y,0,0],$$
$$[0,0,t1,y1],[t1,0,0,x+y],[t1,0,-t1,x+y-y1],[1,0,-1,0].$$

Az összes mátrix rangja egy, azonban ez az egyenlet nem kvázi-lineáris, így a feladat megoldhatóságáról még nem tudunk mit mondani.

4.3. Kompakt halmaz készítése

Az előző fejezet utolsó példájának eredeti egyenletének eredeti alakja

$$\left(f\left(t\left(x+y\right)\right)-f\left(tx\right)\right)\left(f\left(x+y\right)-f\left(y\right)\right) = \left(f\left(t\left(x+y\right)\right)-f\left(ty\right)\right)\left(f\left(x+y\right)-f\left(x\right)\right). \tag{4.3}$$

(Ez az egyenlet Járai [28]-ben szerepel, a szerző nem publikálta.) Ebből algebrai átalakítással kapjuk az előző példa szerinti alakot. Az egyenlet nem kvázi-lineáris, a 2.1.4

alkalmazható. Feladatunk az, hogy a szimbolikus intervallumkezeléssel megtaláljuk azt a C kompakt halmazt, hogy $x \in C$ esetén az összes belső függvény értéke ebbe a halmazba esik. Az így kapott

$$f(x) = \frac{f(y)\left(f\left(t\left(x+y\right)\right) - f\left(tx\right)\right)}{f\left(t'\,x'\right) - f\left(t'\,y'\right)} + \frac{\left(f\left(tx\right) - f\left(ty\right)\right)\left(f\left(x'\right)\left(f\left(t'\ \left(x+y\right)\right) - f\left(t'\,y'\right)\right) - f\left(y'\right)\left(f\left(t'\ \left(x+y\right)\right) - f\left(t'\,x'\right)\right)\right)}{\left(f\left(t'\,x'\right) - f\left(t'\,y'\right)\right)\left(f\left(t\left(x+y\right)\right) - f\left(ty\right)\right)}$$

függvényegyenletben a változók nem-negatívak. Most két utat követhetünk; az első lehetőség az általunk használt komputeralgebrai rendszerek lehtővé teszi, hogy változókra matematikai kitételeket rójunk:

Az „assume”

```
>assume(t::realcons); additionally(t > 0);
>assume(y::realcons); additionally(y > 0);
>assume(t1::realcons); additionally(t1 > 0);
>assume(y1::realcons); additionally(y1 > 0);
>assume(x::realcons); additionally(a <= x and x <= b);
>additionally(x1+y1 = x+y);
```

Az **assume** segítségével az általunk kidolgozott intervallum aritmetika el tudja végezni a szimbolikus kifejezések kezelését. Ezzel a megközelítéssel az a probléma, hogy ha közben szükségünk van egyenletrendszer (vagy egyenlőtlenségek) megoldására, ezeket a beépített megoldási mechanizmusok nem használják.
Így ebben az esetben mi is inkább eltekintünk ettől. Ehelyett ebben a viszonylag egyszerű esetben elegendő egy lineáris rendszert megoldanunk. Ha $a, b \in \mathbb{R}$ nemnegatív valós számok, és a változók az $[a, b]$ zárt intervallumból kerülnek ki, akkor a

```
>constraints := [op({t > 0, t1 > 0, x > 0, y > 0, y1 > 0, x1+y1 = x
    +y, x1 <> y1, y1 < x+y})]
```

feltételekkel, továbbá a

```
>G := [y, y1, t*x, t*y, t*(x+y), t1*y1, t1*(x+y), t1*(x+y-y1), x+y-
    y1]
```

belső függvényekkel az

```
>a, a, a*x, a^2, a*(x+a), a^2, a*(x+a), a*x, x
```

illetve

```
>b, b, b*x, b^2, b*(x+b), b^2, b*(x+b), b*x, x
```

értékeket kapjuk. Ebből a-ra annak kell teljesülnie, hogy

```
>solve(0 < a^2+a and a^2+a < 1, {a})\,.
```

Vagyis $\left\{0 < a, a < 1/2\sqrt{5} - 1/2\right\}$. Hasonló igaz b-re, figyelembe véve az $a < b$ feltételt, a kapott intervallum bármely két számpárja jó.

5. ÖSSZEGZÉS

Ebben a rövid fejezetben összefoglaljuk, hová jutottunk, illetve felvázoljuk a továbblépési lehetőségeket.

5.1. Összefoglalás

Az első fejezetben röviden összefoglaltuk a függvényegyenletek kezelésében és megoldásában használható számítógépes technikákat. Ismertettük Castillo és Iglesias „általános megoldó" programcsomagját, ami MATHEMATICA -ban készült. Ez nem lett sohasem a MATHEMATICA integráns része, ki is fejtettük, véleményünk szerint miért nem tudunk ma még ilyen általános szoftvert készíteni. Speciális esetekre viszont minden további nélkül lehetséges, erre példa ugyanezen fejezet „Megoldás helyettesítések véges csoportjával" alfejezete.
A második fejezetben a regularitás matematikai hátterét ismertettük [29] eredményeinek összefoglalásával. A harmadik fejezetben az általunk kifejlesztett szimbolikus intervallum aritmetikát ismertettük, Czirbusz [20]. A függvényegyenletek algebrai átalakításakor ez az egyik általános módszer, amivel az értelmezési tartományokat illetve értékkészleteket ki tudjuk számolni.
A negyedik fejezetben ismertettük a regularitás problematikájának számítógépes megközelítésével kapcsolatos munkánkat, ez a [19] alapján készült. Ebben a fejezetben két rövid részben ennek továbbviteli lehetőségével foglalkoztunk: egy geometria megközelítést vázoltunk föl, illetve a kompakt halmazok kiszámítására adtunk példát.
Az értekezés során elkészített programok a MAPLE illetve a SAGE komputeralgebra rendszerekben készültek. Ez nem jelent lényeges korlátozást, ugyanakkor a matematikai műveletek végrehajtását nagymértékben egyszerűsíti.

5. ÖSSZEGZÉS

5.2. Továbbfejleszés

Az első fejezetben ismertetett csoportelméleti megfontolásokhoz hasonló módszerekkel függvényegyenletek széles skálájára lehet megoldó programot írni. Természetesen figyelembe kell venni, hogy az esetek jelentős részében a gyakorlati alkalmazásokban a reguláris megoldások keresésének van nagyobb jelentősége. Mivel a belső függvények többnyire egyszerűek - lineáris kifejezések, első- vagy másodfokú kifejezések hányadosai, ebben az irányban viszonylag egyszerűen lehet továbbhaladni. Az intervallum aritmetikában meg kell oldani, hogy többváltozós esetben is ki tudja számítani a megfelelő értelmezési tartományokat illetve értékkészleteket. Távlatilag célszerű a függvényegyenletek kezelését is objektumok segítségével átírni, erre a SAGE alkalmasabbnak látszik, mivel az alapul szolgáló Python programozási környezetben ez természetes.

6. SUMMARY

In this short chapter we summarize the road we worked over, and draw up the possibilities of further improvements.

6.1. Summary

In the first chapter we presented the computational methods suitable for handling and solving functional equatzions. We described the „general solver" MATHEMATICA packages written by Castillo and Iglesias. This package never does to be integrant part of MATHEMATICA integráns része, we articulated our opinion about the impossibility of that program package. Naturally for special forms if equation it works without any difficulty. An example for this case is the part „Solving functional equations via groups of finite substitutions " of this chapter.
In the second chapter the mathematical background of regurarity was described with summarizing the results from [29]. In the third chapter was summarized the „Symbolic interval manipulation" developed by the author of this tehesis bya Czirbusz [20]. This method serves under the algebraic transformations of functional eqautions to compute the new domains and ranges.
In the fourth chapter was presented the computational works of author on the checking the regularity theorems, this happened by foundations of [19]. In this chapter was dealt with thwo short sections to further considerations. One of them is a geometric approach, the second one is a method for computing , compact sets.
In the dissertation the programs was made by MAPLE and SAGE computer algebraic systems. This is not a restriction, while teh execution of mathematical operations greatly simplified.

6.2. Possible further improvements

Similar methots to the group theoretic considerations mentioned in the first chapter allows to make programs to solve wide range of functional equations. Naturally we must take into consideration that in practical applications more importants the equations with some regualrity. Because the inner functions mostly are simply - linear expressions, quotients of first or second order polynomials - it seems simply to forward further. in the symbolic intervallum manipulation we must solve the problem of functional equations in several variables(domains and ranges). It seems expedient to rewrite tha handling of functionals equations with object, for this the SAGE seems to be more suitable because of the underlying Python environment.

A. PROGRAMKÓDOK

A.1. A számítógépes alkalmazások áttekintése

A.1.1. Sorfejtések

A.1.2. Közepek invarianciaegyenlete

Közepek invarianciaegyenlete

```
>G := (p, q, x, y)-> ((x^p + y^p)/(x^q + y^q))^(1/(p-q)) :
>F := x -> G(p, q, G(a, b, exp(x), exp(-x)), G(c, d, exp(x), exp(-x)))/G(p, q, exp(x), exp(-x)) :
>eq1 := simplify(coeftayl(F(x), x = 0, 2))
       eq1 := 1/4 c + 1/4 a + 1/4 b - 1/2 p - 1/2 q + 1/4 d
>sb1 := a = w+v+sqrt(r+s);
       a=w+v+sqrt(r+s)
>sb2 := b = w+v-sqrt(r+s):
>sb3 := c = w-v+sqrt(r-s):
>sb4 := d = w-v-sqrt(r-s):
>sb5 := p = w+sqrt(t):
>sb6 := q = w-sqrt(t):
>H := x -> subs(sb1, sb2, sb3, sb4, sb5, sb6, F(x)):
>eq2 := simplify(coeftayl(H(x), x = 0, 4))
       eq2 := -1/3 wr - 1/3 sv + 1/3 tw
>eq3 := simplify(coeftayl(H(x), x = 0, 6))
       eq3:= -2/3 v^2 w^3 + 2/3 r v^2 w + 4/9 r w^3 + 4/9 s v^3 + 2/3 s v w^2 - 4/9 t w^3 + 2/15 r^2 w + 4/15 r s v + 2/15 s^2 w - 2/15 t^2 w
>eq4 := simplify(coeftayl(H(x), x = 0, 8)):
>eq5 := simplify(coeftayl(H(x), x = 0, 10)):
>eq6 := simplify(coeftayl(H(x), x = 0, 12)):
>EQ := [eq2, eq3, eq4, eq5, eq6]:
>with(Groebner):
>B := Basis(EQ, plex(r, s, t, v, w)):
>solve(B)
       {r = r, s = s, t = t, v = 0, w = 0}, {r = t, s = 0, t = t, v = 0, w = w},
       {r = r, s = 0, t = t, v = v, w = 0}, {r = w^2, s = 0, t = w^2, v = v, w = w},
```

$$\{r = r, s = w^2 - r, t = w^2, v = w, w = w\}, \{r = r, s = -w^2 + r, t = w^2, v = -w, w = w\}$$

A.1.3. Megoldás helyettesítések véges csoportjával

A.1.3.1. Csoport létrehozása Dimino-algoritmussal

A.1 Program: Dimino Algoritmus

```
# Routines for listing all the group elemets
id(x) = x
def OP(f,g):
    return f(g(x)).simplify_full().function(x)

def Dimino(S):
    """
    INPUT:
    - ``S`` the list of generators
    OUTPUT:
    - ``elements`` the list of elements of generated group
    We  use this procedure to generating group of functions.

    EXAMPLES::
        sage: f2(x) = (x-1)/(x+1)
        sage: f3(x) = -1/x
        sage: f4(x)= (x+1)/(1-x)
        sage: Dimino([f2, f3, f4])
        [x |--> x, x |--> (x - 1)/(x + 1),
         x |--> -1/x, x |--> -(x +1)/(x - 1)]
    """

    t = len(S)
    # The unit element always in the group
    Ord = 0; elements = [id]
    g = S[0];
    # the first, cyclic subgroup
    while g(x)<>x:
        Ord +=1; elements.append(g)
        g = OP(g, S[0])

    # building the group inductively
    for i in xrange(2, t):
        if not S[i] in elements: # the next element is not redundant
```

```
            prevord = Ord
            # add the coset
            Ord += 1; elements.append(S[i])
            for j in xrange(2, prevord):
                Ord +=1; elements.append(OP(elements[j], S[i]))
            # the position of representative
            reppos = prevord + 1
            while true:
                for s in S:
                    elt = OP(elements[reppos], s)
                    if not elt in elements:
                        Ord +=1; elements.append(elt)
                    for j in xrange(2, prevord):
                        Ord += 1; elements.append(OP(elements[j], elt))
            # the next coset representative
            reppos += prevord
            if reppos > Ord:
                break

    return elements
```

A.1.3.2. A permutációcsoport elkészítése

A.2 Program: Csoportreprezentációk

```
# Routines for regular representation

def delta(G, k, j, i):
    """
    The "Kronecker-delta" for the group G.

\author{}
    INPUT:
    - ``G`` - the list of group-elements
    - ``k,j,i`` - index of group elements

    OUTPUT:
    1 if G[k]*G[j] = G[i]
    0 otherwise
    """
    if G.index(OP(G[k],G[j])) == i:
        return 1
    else:
        return 0
```

```
def RegRep(G):
    """
    Regular matrix representation of a finite group

    INPUT:
    - ``G`` - the list of group-elements

    OUTPUT
    - ``M`` - a list of quadratic permutation matrices

    M[k][i,j] = delta(G,k,j,i)
    """
    n = len(G)
    M = [identity_matrix(n)]
    for k in xrange(1, n) :
        A = matrix(n)
        for i in xrange(0, n):
            for j in xrange(0, n):
                 A[i, j] = delta(G, k, j, i)
        M.append(A)
    return M

def ToPerm(M, cyclic=false):
    """
    Convert a permutation matrix to permutation; if the cyclic flag is true,
    then it makes the cyclic product, otherwise in two line notation.
    INPUT:
    - ``M`` - a permutation matrix
    - ``cyclic`` - a flag

    OUTPUT:
    - ``P`` - the permutation

    """
    n = M.nrows()
    if cyclic:
        Ndx = [1] * n
        first = 1
        P = []
        while first < n:
            Ndx[first - 1] = 0
            C = [first]
            a = M[first - 1,:].list()
            while true:
                j = a.index(1) + 1
                if j == first:
                    break
                else:
```

```
                    C.append(j)
                    Ndx[j - 1] = 0
                    a = M[j - 1,:].list()
            P.append(tuple(C))
            try:
                first = Ndx.index(1) + 1
            except:
                first = n + 1
        return P

    else:
        P = matrix(2, n)
        P[0,:]=[[i for i in xrange(1, n + 1)]]
        for i in xrange(0, n):
            a = M.row(i).list()
            P[1,i] = a.index(1) + 1
        return P

def ToPerms(M, cyclic=false):
    """
    Convert a list of permutation matrices to permutations
    - ``M`` - list of  permutation matrices
    - ``cyclic`` - a flag

    OUTPUT:
    - ``P`` - the list of permutations

    """
    P = []
    for m in M:
        P.append(ToPerm(m, cyclic))
    return P
```

A.2. Szimbolikus intervallum aritmetika

A.2.1. Az intervallum objektum

A.3 Program: Az intervallum objektum

```
Interval := proc (a, b, lO := true, rO := true)
  module ()
    export Left, Right, LeftOpen, RightOpen, Radius, Width,
           MidPoint, Magnitude, Closed, Open, ModulePrint,ModuleApply, isInterval;
    if is(b < a) then
      Left := b; Right := a
    else
      Left := a; Right := b
    end if;
    LeftOpen := lO; RightOpen := rO;
    Width := abs(Left-Right);
    Radius := (1/2)*Width;
    MidPoint := (1/2)*Left+(1/2)*Right;
    Magnitude := max(abs(Left), abs(Right));
    Open := LeftOpen and RightOpen;
    Closed := `not`(LeftOpen or RightOpen);
    isInterval:=true;
    ModulePrint := proc ()
      local s;
      s := "";
      if LeftOpen then s := cat(s, "(")
      else s := cat(s, "[")
      end if;
      s := cat(s, convert(Left, string), ",", convert(Right, string));
      if RightOpen then s := cat(s, ")")
      else s := cat(s, "]")
      end if;
    end proc ;
    ModuleApply := proc()
      thismodule:-ModulePrint()
    end proc
  end module
end proc:
```

A.2.2. Intervallumok szorzása

A.4 Program: Intervallumszorzás

```
`*`:= overload([
  proc(A::NotInterval,B::Interval)
    option overload;
    if is(A > 0) then
      Interval(B:-Left*A, B:-Right*A, B:-LeftOpen, B:-RightOpen)
    else
      Interval(B:-Right*A, B:-Left*A, B:-RightOpen, B:-LeftOpen)
    end if
  end proc,
  proc(A::Interval, B::`**`)
    option overload;
    A * Reciprocial(op(1,B))
  end proc,
  proc(A::Interval,B::NotInterval)
    option overload;
    if is(B > 0) then
      Interval(A:-Left*B, A:-Right*B, A:-LeftOpen, A:-RightOpen)
    else
      Interval(A:-Right*B, A:-Left*B, A:-RightOpen, A:-LeftOpen)
    end if
  end proc,
  proc (A::Interval, B::Interval)
    local a1, a2, b1, b2, c1, c2, la, ra, lb, rb, lc,
      rc, a01, b01, x, y;
    a1 := A:-Left; a2 := A:-Right; la := A:-LeftOpen;
    ra := A:-RightOpen;
    b1 := B:-Left; b2 := B:-Right; lb := B:-LeftOpen;
    rb := B:-RightOpen;
    if is(0 <= a1) and is(0 <= b1) then
      c1 := a1*b1; c2 := a2*b2;
      lc := la or lb; rc := ra or rb
    elif is(a1 < 0) and is(0 < a2) and is(0 <= b1) then
      c1 := a1*b2; c2 := a2*b2;
      lc := la or rb; rc := ra or rb
    elif is(a2 <= 0) and is(0 <= b1) then
      c1 := a1*b2; c2 := a2*b1;
      lc := la or rb; rc := ra or lb
    elif is(0 <= a1) and is(b1 < 0) and is(0 < b2) then
      c1 := a2*b1; c2 := a2*b2;
      lc := ra or lb; rc := ra or rb
    elif is(0 <= a2) and is(b1 < 0) and is(0 < b2) then
      c1 := a1*b2; c2 := a1*b1;
```

```
      lc := la or rb; rc := la or lb
    elif is(0 <= a1) and is(b2 <= 0) then
      c1 := a2*b1; c2 := a1*b2;
      lc := ra or lb; rc := la or rb
    elif is(a1 < 0) and is(0 < a2) and is(b2 <= 0) then
      c1 := a2*b1; c2 := a1*b1;
      lc := ra or lb; rc := la or lb
    elif is(a2 <= 0) and is(b2 <= 0) then
      c1 := a2*b2; c2 := a1*b1;
      lc := ra or rb; rc := la or rb
    else #0<=a2<a01 and 0<=b01<b2
      a01 := -a1; b01 := -b1;
      if is(0 <= a01) and is(a01 <= a2) and
         is(0 <= b01) and is(b01 <= b2) then
        c2 := a2*b2; rc := ra or rb; x := a1*b2; y := a2*b1;
        if is(x <= y) then c1 := x; lc := la or rb
        else c1 := y; lc := ra or lb
        end if
      elif is(0 <= a2) and is(a2 <= a01) and
           is(0 <= b2) and is(b2 <= b01) then
        c2 := a1*b1; rc := la or lb; x := a1*b2; y := a2*b1;
        if is(x <= y) then c1 := x; lc := la or rb
        else c1 := y; lc := ra or lb
        end if
      elif is(0 <= a01) and is(a01 <= a2) and
           is(0 <= b2) and is(b2 <= b01) then
        c1 := a2*b1; lc := ra or lb; x := a1*b1; y := a2*b2;
        if is(y <= x) then c2 := x; rc := la or lb
        else c2 := y; ra := ra or rb
        end if
      else
        c1 := a1*b2; lc := la or rb; x := a1*b1; y := a2*b2;
        if is(y <= x) then c2 := x; rc := la or lb
        else c2 := y; ra := ra or rb
        end if ;
      end if ;
    end if;
    Interval(c1, c2, lc, rc)
  end proc
]);
```

A.2.3. Intervallumok SAGE -ben

A.5 Program: Intervallumok SAGE -ben

```
class Interval:
    def __init__(self, a, b, lo, ro):
        if b < a:
            self.left = b; self.right = a
        else:
            self.left = a; self.right = b
        self.left_open = lo; self.right_open = ro
    def width(self):
        return abs(self.left - self.right)
    def radius(self):
        return self.width() / 2
    def midpoint(self):
        return (self.left+self.right) / 2
    def magnitude(self):
        return max(abs(self.left), abs(self.right))
    def open(self):
        return (self.left_open and self.right_open)
    def closed(self):
        return not(self.left_open or self.right_open)
    def __repr__(self):
        s = ''
        if self.left_open:
            s = s + '('
        else:
            s = s + '['
        if self.left == Infinity:
            s = s + 'Infinity'
        elif self.left == -Infinity:
            s = s + '-Infinity'
        else:
            s = s + self.left.str()
        s = s + ','
        if self.right == Infinity:
            s = s + 'Infinity'
        elif self.right == -Infinity:
            s = s + '-Infinity'
        else:
            s = s + self.right.str()
        if self.right_open:
            s = s + ')'
        else:
            s = s + ']'
```

```
        return s
    def __str__(self):
        return self.__repr__()
    def __add__(self, other):
        if other.__class__.__name__ == 'Interval':
            return Interval(self.left+other.left, self.right+ other.right,
                            self.left_open or other.left_open,
                            self.right_open or other.right_open)
        else:
            return Interval(self.left+other, self.right+other,
                            self.left_open, self.right_open)
    def __radd__(self,other):
        if isinstance(other, Interval):
            return Interval(self.left+other.left, self.right+ other.right,
                            self.left_open or other.left_open,
                            self.right_open or other.right_open)
        else:
            return Interval(self.left+other, self.right+other, self.left_open,
                            self.right_open)
    def __sub__(self, other):
        if isinstance(other, Interval):
            return Interval(self.left-other.left, self.right-other.right,
                            self.left_open or other.left_open,
                            self.right_open or other.right_open)
        else:
            return Interval(self.left-other, self.right-other, self.left_open,
                            self.right_open)
    def __mul__(self, other):
        if isinstance(other, Interval):
            a1 = self.left; a2 = self.right;
            la = self.left_open;
            ra = self.right_open;
            b1 = other.left; b2 = other.right;
            lb = other.left_open;
            rb = other.right_open;
            if 0 <= a1 and 0 <= b1:
                c1 = a1*b1; c2 = a2*b2;
                lc = la or lb; rc = ra or rb
            elif a1 < 0 and 0 < a2 and 0 <= b1:
                c1 = a1*b2; c2 = a2*b2;
                lc = la or rb; rc = ra or rb
            elif a2 <= 0 and 0 <= b1:
                c1 = a1*b2; c2 = a2*b1;
                lc = la or rb; rc = ra or lb
            elif 0 <= a1 and b1 < 0 and 0 < b2:
                c1 = a2*b1; c2 = a2*b2;
                lc = ra or lb; rc = ra or rb
            elif 0 <= a2 and b1 < 0 and 0 < b2:
                c1 = a1*b2; c2 = a1*b1;
```

```
            lc = la or rb; rc = la or lb
        elif 0 <= a1 and b2 <= 0:
            c1 = a2*b1; c2 = a1*b2;
            lc = ra or lb; rc = la or rb
        elif a1 < 0 and 0 < a2 and b2 <= 0:
            c1 = a2*b1; c2 = a1*b1;
            lc = ra or lb; rc = la or lb
        elif a2 <= 0 and b2 <= 0:
            c1 = a2*b2; c2 = a1*b1;
            lc = ra or rb; rc = la or rb
        else: #0<=a2<a01 and 0<=b01<b2
            a01 = -a1; b01 = -b1;
            if 0 <= a01 and a01 <= a2 and 0 <= b01 and b01 <= b2:
                c2 = a2*b2; rc = ra or rb; x = a1*b2; y = a2*b1;
                if x <= y:
                    c1 = x; lc = la or rb
                else:
                    c1 = y; lc = ra or lb
            elif 0 <= a2 and a2 <= a01 and 0 <= b2 and b2 <= b01:
                c2 = a1*b1; rc = la or lb; x = a1*b2; y = a2*b1;
                if x <= y:
                    c1 = x; lc = la or rb
                else:
                    c1 = y; lc = ra or lb
            elif 0 <= a01 and a01 <= a2 and 0 <= b2 and b2 <= b01:
                c1 = a2*b1; lc = ra or lb; x = a1*b1; y = a2*b2;
                if y <= x:
                    c2 = x; rc = la or lb
                else:
                    c2 = y; ra = ra or rb
            else:
                c1 = a1*b2; lc = la or rb; x = a1*b1; y = a2*b2;
                if y <= x:
                    c2 = x; rc = la or lb
                else:
                    c2 = y; ra = ra or rb
        return Interval(c1, c2, lc, rc)
    else:
        if other >= 0:
            return Interval(self.left * other, self.right*other, self.left_open,
                self.right_open)
        else:
            return Interval(self.right * other, self.left*other, self.right_open,
                 self.left_open)

def __rmul__(self, other):
    if not isinstance(other, Interval):
        if other >= 0:
            return Interval(self.left * other, self.right*other,
```

```
                            self.left_open, self.right_open)
        else:
            return Interval(self.right * other, self.left*other,
                            self.right_open, self.left_open)

def __div__(self, other):
    if isinstance(other, Interval):
        if other.left > 0 or other.right <0:
            return self  * other.reciprocial()
        elif self.left <=0 and 0 <=self.right:
            return Interval(-infinity,infinity, true,true)
        elif (self.left >0 or self.right <0) and (other.left == 0 and other.right
             == 0):
            return []
        elif self.right < 0 and self.left < self.right and self.right == 0:
            return Interval(self.right/other.left,infinity,self.right_open or
                other.left_open, true)
        elif self.right < 0 and other.left<0 and 0 < other.right:
            return [Interval(-infinity, self.right/other.right,true, self.
                right_open or other.right_open),
                    Interval(self.right/other.left,infinity, self.right_open or
                        other.left_open, true)]
        elif self.right < 0 and 0 == other.left and other.left < other.right:
            return Interval(-infinity, self.right/other.Right,true, self.
                right_open or other.right_open)
        elif self.left >0 and self.left < self.right and self.right == 0:
            return Interval(-infinity,self.left/other.left,true, self.left_open
                or other.left_open)
        elif self.left >0 and other.left<0  and 0 < other.right:
            return [Interval(-infinity, self.left/other.left,true,self.left_open
                or other.right_open),
                    Interval(self.left/other.right, infinity, self.left_open or
                        other.right_open,true)]
        else: # (A:-Left >0) and (0 = B:-Left) and (B:-Left<B:-Right)
            return null

def __rdiv__(self, other):
    if not isinstance(other, Interval):
        self.reciprocial() * other


def reciprocial(self):
    if self.left > 0 or self.right < 0:
        return Interval(1/self.right, 1/self.left, self.right_open, self.
            left_open)
    else:
        return null
```

A.3. A regularitás vizsgálata

A.3.1. Kvázi lineáris egyenletek

A.3.1.1. A főprogam

A.6 Program: A rangkritérium ellenőrzésének főprogramja

```
FECheck := proc (FE, S := [-infinity < XS and XS < infinity,-infinity < YS and YS <
    infinity])
 #S: list of constrainst (inequalities)
 local sol, G, DG, SR, dg, BP, p, Elimed, msg;
 global FU, FO, FIN, XS, YS, j;
 msg := "Sorry. The current version of the program does not
        support this type of equations.";
 if not assigned(FU) then
   NewTask()
 end if;
 #G: the inner functions
 #DG: the derivatives of g-s
 #BP: the bad places
 G := []; DG := {}; BP := {};
 Elimed:=false;
 # Try eliminating f(x).
 for j from 0 to 2 do
   sol := Elimf(FE, S, j);
   if is(sol=NULL) then
     next
   end if;
   G := Parse(rhs(sol[1][1]));
   if evalb(sol <> NULL) then
     Elimed := andmap(depends, G, YS)
   end if;
   if Elimed then
     break
   end if
 end do;
 if not Elimed then
   error "I can not eliminate f(x) from the equation"
 elif evalb(sol <> NULL) then
    #checking the conditions of Theorem 1.29
   if nops(G) = 2 and G[1][1] <> YS and G[2][1] <> YS then
     error msg
```

```
    else
     #checking the conditions of Theorem 1.30
     if Error130(rhs(sol[1][1])) then
       error msg
     end if
    end if ;
    for p in G do
      if nops(p) = 1 then
        DG := `union`(DG, {diff(p, YS)})
      end if;
      if nops(p) = 2 then
        DG := `union`(DG, {[diff(p[1], YS), p[2]]})
      end if
    end do;
    for dg in DG do
      p := CheckRank(dg, CNT);
      if p <> [] and p <> {} then
        BP := `union`(BP, {p})
      end if
    end do;
    return BP
  end if;
  return BP
end proc:
```

A.3.1.2. Kvázi-megoldó

A.7 Program: A pszeudo-megoldó kódja

```
Elimf := proc (FE,
               S := [-infinity < XS and XS < infinity,
                     -infinity < YS and YS < infinity],
               Mode := 0)
 local EQ, EQ1, P, PP, sol, p, TEMP, pp, pt, T1, T2, T3, T4, T5;
 global FU, FO, FIN, XS, YS, CNT;
 if not assigned(FU) then
   SetSymbols()
 end if;
 if not assigned(CNT) then
   CNT := ConvertToInterval(S)
 end if;
 T1 := FU(XS); T2 := FU(XS*YS);
 T3 := FU(XS/YS); T4 := FU(YS/XS);
 T5 := XS; EQ := lhs(FE)-rhs(FE);
 P := [op(EQ)];
 PP := zip(`*`, P, map(sign, P));
 sol := {};
 if Mode < 1 then
   if evalb(`in`(T1, PP)) then
     sol := solve(EQ, {T1});
     return [sol, [op(rhs(sol[1]))], EQ]
   end if
 end if;
 if Mode < 2 then
   for p in PP do
     if type(p, function) and nops([op(p)]) < 2 then
       if type(op(p), `+`) then
         EQ1 := TEMP = op(p);
         sol := solve(EQ1, {T5});
         EQ := subs(TEMP = T5,
                    subs(`union`({op(p) = TEMP}, sol), FE));
         CNT[TEMP] := IntervalAdd(CNT[XS], CNT[YS]);
         CNT[XS] := IntervalSubtract(CNT[TEMP], CNT[YS]);
         unassign('CNT[TEMP]');
         return Elimf(EQ)
       end if
     end if
   end do
 end if;
 pp := [];
 for p in PP do
```

```
    if type(p, function) and nops([op(p)]) < 2 then
      if type(op(p), `*`) then
        pp := p;
        if evalb(op(p) = XS/YS) or evalb(op(p) = YS/XS) then
          EQ1 := TEMP = op(p);
          sol := solve(EQ1, {T5});
          EQ := subs(TEMP = T5,
                     subs(`union`({op(p) = TEMP}, sol), FE));
          CNT[TEMP] := IntervalMult(CNT[XS], CNT[YS]);
          CNT[XS] := IntervalDiv(CNT[TEMP], CNT[YS]);
          unassign('CNT[TEMP]');
          return Elimf(EQ)
        end if
      end if
    end if
  end do;
  if pp <> [] then
    p := pp; EQ1 := TEMP = op(p);
    sol := solve(EQ1, {T5});
    EQ := subs(TEMP = XS, subs(`union`({op(p) = TEMP}, sol), FE));
    return Elimf(EQ)
  end if
end proc;
```

A.3.1.3. A parser

A.8 Program: A parser kódja

```
Parse := proc (EQ)
 local EQS, i, j, bc, n, m, F, tmp, c, p, pp, EQF;
 global FU;
 if not assigned(FU) then
   SetSymbols()
 end if;
 bc := 0; F := []; EQF := sign(EQ)*EQ;
 if Operator(EQF) or op(0, EQF) = evaln(FU) then
   EQS := convert(EQF, string);
   n := length(EQS);
   m := 1;
   while m <= n do i := searchtext(cat(FU, "("), EQS, m .. -1);
   if i = 0 then
     return F
   else
     bc := 1; j := m+i+1; tmp := "";
     while 0 < bc and j <= n do
       c := EQS[j];
       if c = "(" then
         bc := bc+1
       elif c = ")"
       then
         bc := bc-1
       else
       end if;
       j := j+1
     end do;
     if 2 <= j-i then
       tmp := substring(EQS, m+i .. j-1);
       F := [op(F), [parse(tmp)]]
     end if
   end if;
   m := j+1
 end do
 elif type(EQF, function) then
   if op(0, EQF) = 'sum' or op(0, EQF) = 'product' then
     pp := [Parse(op(1, EQF)), op(2, EQF)];
     F := [op(F), ListTools[FlattenOnce](pp)]
   else
     for p in op(EQ) do
       pp := Parse(p);
       if pp <> [] then
```

```
          F := [op(F), ListTools[FlattenOnce](pp)]
        end if
      end do
    end if
    else
  end if;
  return F
end proc;
```

A.3.1.4. A rangkritérium ellenőrzése

A.9 Program: A rang ellenőrzése

```
CheckRank := proc (M, S := [])
 local MM, dm, m, p, BP, sol, EQ, nn, i;
 global FU, FO, FIN, XS, YS;
 if not assigned(FU) then
   SetSymbols()
 end if;
 EQ := {};
 L := [indices(S)];
 for l in L do
   if CNT[l[1]] <> real then
     EQ := `union`(EQ, {convert(`in`(l[1], CNT[l(1)]), `relation`)})
   end if
 end do
 BP := {};
 for m in M do
   MM := Matrix(M);
   dm := [LinearAlgebra[Dimension](MM)];
   if dm[1] = dm[2] and dm[2] = 1 then
     if type(M[1], numeric) then
       if M[1] = 0 then
         BP := `union`(BP, {M[1]})
       end if
     else
       EQ := `union`(EQ, {m});
       sol := solve(EQ, {XS, YS});
       nn := nops([sol]);
       if 1 < nn then
         for i to nn do
           if sol[i] <> {} then
             BP := `union`(BP, {sol[i]})
           end if
         end do
       else
         BP := `union`(BP, sol)
       end if
     end if
   else
   end if
 end do;
 return BP
end proc:
```

JELÖLÉSEK

$\underline{X}, \overline{X}$ Az X intervallum bal– és jobb végpontjai

$\mathbb{C}, \mathbb{R}, \mathbb{Q}, \mathbb{Z}, \mathbb{N}$ a komplex, a valós, a racionális, az egész és a természetes számok halmaza

$\mathbb{R}_+$ a pozitív valós számok halmaza

$f \circ g$ az f és g függvények kompozíciója

$f : A \to B$ függvény megadása

i, j, k, m, n természetes számok

$X \odot Y$ valamilyen művelet az X és Y intervallumokkal

IRODALOMJEGYZÉK

[1] Asksage. URL http://ask.sagemath.org. 22

[2] *MPFI, a multiple precision interval arithmetic library based on MPFR*. 35

[3] *Arbitrary Precision Real Intervals*. 35

[4] *A Package for Symbolic Solution of Functional Equations*, 1995. 14

[5] J Aczél. *Lectures on Functional Equations and Their Applications*. Academic Press, 1966. 1, 45

[6] J. Aczél and Z. Daróczy. *On Meausures of Information and Their Characterisation*. Mathematics in Science and Engineering. Academic Press, 1975. 45

[7] Sz. Baják and Zs. Páles. Computer aided solution of invariance equation for two-variable stolarsky means. *Applied Mathemetics and Computation*, 216, 2010. 12

[8] Sz. Baják and Zs. Páles. Computer aided solution of the invariance equation for two-variable stolarsky means. *Applied Mathematics and Computation*, 216: 3219–3227, 2010. ISSN 0096-3003. 12

[9] Sz. Baják and Zs. Páles. Solving invariance equations involving homogeneous means with the help of computer. *Applied Mathematics and Computation*, 219: 6297–6315, 2013. ISSN 0096-3003. 12

[10] M. Bessenyei. Functional equations and finite groups of substitutions. *The American Mathematical Monthly*, 117(10):921–927, 2010. 15, 18, 23

[11] M. Bessenyei and Cs. G. Kézi. Functional equations and group substitutions. *Linear Algebra and its Applications*, 434, 2011. 15, 18, 23

[12] M. Bessenyei and Cs. G. Kézi. Solving functional equations via finite substitutions. *Aequationes Mathematicae*, 85:593–600, 2013. 15, 18

[13] M. Bessenyei, G. Horváth, and Cs. G. Kézi. Functional equations on finite groups of substitutions. *Expo. Math.*, 30:283–294, 2012. 15, 18, 23

[14] Jonathan .M Borwein and Peter B. Borwein. *Pi and the AGM*. John Wiley & Sons, 1987. 12

[15] V.S. Brodskii and A.K. Slipenko. *Functional Equations*. Visa Skola, Kiev, USSR, 1983. 16

[16] G. Butler. Fundamental algorithms for permutation groups. 1991. 20

[17] E. Castillo and A. Iglésias. A package for symbolic solution of functional equations of real variables. *Aequationes Mathematicae*, 54:181–198, 1997. 14

[18] Interval Computations. Interval and related software. "http://www.cs.utep.edu/interval-comp/intsoft.html". 33

[19] S Czirbusz. Testing regularity of functional equations with computer. *Aequationes Mathematicae*. 43, 51, 53

[20] S Czirbusz. Symbolic interval manipulation. *Annales Univ. Sci. Budapest., Sect. Comp*, 40, 2013. 40, 51, 53

[21] Z. Daróczy. Notwendige und hinreichende bedingungen fiir die existenz von nichtkonstanten lösungen linearen funktionalgleichungen. *Acta Sci. Math. Szeged*, 22:31–41, 1961. 1

[22] Costas J Efthimiou. *Introduction to Functional Equations: Theory and problem-solving strategies for mathematical competitions and beyond*. MSRI Mathematical Circles Library. 1

[23] E. Hansen and G. W. Walster. *Global Optimization Using Interval Analysis*. Marcel Dekker inc., 2004. 29

[24] T. Hickey, Q. Ju, and M.H. van Emden. Interval arithmetic: from principles to implementation. *Journal of the ACM*, 48(5):1038–1068, September 2001. 29

[25] A. Házy. Solving linear two variable functional equation with computer. *Aequationes Mathematicae*, 67:47–62, 2004. 11

[26] Luc Jaulin and G Chabert. Resolution of nonlinear interval problems using symbolic interval arithmetic. *Engineering Applications of Artificial Intelligence*, 23 (6):1035–1040, 2010. 29

[27] Luc Jaulin, Michel Kieffer, Oliver Didrit, and Éric Walter. *Applied Interval Analysis*. Springer, 2001. 29

[28] A. Járai. Checking regularity of a functional equation. 48

[29] A. Járai. *Regularity Propoerties of Functional Equations in Several Varibales*. Advances in Mathematics. Springer, 2005. 2, 4, 5, 7, 25, 26, 28, 45, 51, 53

[30] W. Krämer and W. Hofschuster. intpakx – verified numerics meets computer algebra. `http://www2.math.uni-wuppertal.de/~xsc/software/intpakX/`. 35

[31] M. Kuczma. A survey of the theory of functional equations. *Univ. Beogard. Publ. Elektrotehn. Fak. Ser. Mat. Fiz*, 130:1–64, 1964. 1

[32] Ulrich W. Kulisch and Universität Karlsruhe. Complete interval arithmetic and its implementation on the computer. 29, 37

[33] R. E Moore. *Interval Analysis*. Prentice Hall Inc., Englewood Cliffs, 1966. 29

[34] R. E Moore. *Methods and Applications of Interval Analysis*. SIAM, 1979. 29

[35] R. E Moore, R.B Kearfott, and M.J Cloud. *Introduction to Interval Analysis*. SIAM, 2009. 32, 33, 37

[36] Arnold Neumaier. *Interval methods for systems of equations*. Encyclopedia of Mathematics and its Application. Cambridge University Press, 1990. 29

[37] Zs. Páles. On reduction of linear two variable functional equations to differential equations without substitutions. *Aequationes Mathematicae*, 43:236–247, 1992. 9, 10, 11

[38] C.G. Small. *Functional Equations and How to Solve Them*. Problem Books in Mathematics. Springer, 2007. 1, 16