Eötvös Loránd Tudományegyetem
Természettudományi Kar

Egy polinom-differenciálegyenlet megoldása iterációval

BSc Szakdolgozat

Írta:

Béres Ferenc

Alkalmazott matematikus szakirány

Témavezető:

Sigray István

műszaki gazdasági tanár

Analízis Tanszék

Budapest, 2013

Tartalomjegyzék

Köszönetnyilvánítás

Elsősorban köszönettel tartozom Sigray István egyetemi tanáromnak a korábbi félévek analízis, és komplex függvénytan tárgyainak előadójaként és gyakorlatvezetőjeként, de különösen szakdolgozatom témavezetőjeként végzett munkájáért. Részletes magyarázataival és hasznos útmutatásaival sokat segített a szakdolgozat elméleti felépítésében.

Köszönettel tartozom Lócsi Levente numerikus matematikai programcsomagok gyakorlatvezetőmnek, hogy időt szakított a szakdolgozathoz készített programok átnézésére, és számos hasznos tanáccsal látott el a módszerek implementálása során.

Szeretnék köszönetet mondani minden tanáromnak, akik a három év alatt hozzájárultak a szakmai fejlődésemhez, illetve családomnak akik az eddigi évek alatt mindig kitartóan támogattak.

Bevezető

A differenciálegyenletek vizsgálata, megoldása az alkalmazott matematika mindennapokhoz egyik legközelebb eső ága, hiszen ezekkel írhatók le bizonyos fizikai, kémiai jelenségek, illetve pénzügyi folyamatok. A dolgozatban vizsgált polinom-differenciálegyenlet eltér a megszokott differenciálegyenletektől, hiszen két ismeretlen polinomra adott a differenciálegyenlet.

Az első fejezetben ismertetem a vizsgált polinom-differenciálegyenletet, melynek általánosítását a Jacobi-sejtés motiválja. Az 1.1-es alfejezetben két általános megoldás felírására kerül sor, melyekről mind a valós, mind a komplex esetben bebizonyítom, hogy valóban megoldásai a vizsgált polinom-differenciálegyenletnek. Az 1.2-es alfejezetben a probléma közönséges differenciálegyenlettel való vizsgálatával bemutatom, hogy a polinom-differenciálegyenlet megoldásai a meghatározott esetben hogyan használhatók fel az elemi primitív függvény meghatározására. Ezután nemlineáris egyenletrendszerre vezetem vissza a problémát (1.3-as alfejezet). Célom a megoldások numerikus meghatározása a másod-, harmad-, és negyedfokú esetben egy konstans értékének rögzítése mellett.

Az egyenletrendszer numerikus megoldására a Newton–Raphson-módszert választom. A második fejezetben a módszer konvergencia tulajdonságait, és egy lehetséges globális javítását, a folytatás módszerét, részletezem vektormezőkre. A Broyden-módszerről is említést teszek, amely a Jacobi-mátrix egy meghatározott közelítésével végzi az iterációs lépéseket.

A harmadik fejezetben az általam a *Matlab* numerikus matematikai programcsomagban készített *Equation_sys_solver*, *Equation_sys_solver_tester* és *tesztgrafikon* m-fájlokkal keresem a polinom-differenciálegyenlet megoldásait, illetve vizsgálom a megoldás megtalálásáig megtett iterációs lépések számát a kezdeti közelítés normájának függvényében a második fejezetben említett Newton–Raphson- és Broyden-módszerekre egyaránt. A fejezet végén egy külön példán keresztül a folytatás módszerének egyik jó tulajdonsága is bemutatásra kerül.

A Függelékben egy rövid leírás található az általam készített m-fájlok paramétereiről, implementálásáról, és használatáról. A programhoz szükséges m-fájlok megtalálhatóak a *http://www.cs.elte.hu/~ferdzso/documents/EquSysSolver.zip* tárhelyen. A harmadik fejezetben vizsgált példák tesztelését, a programok letöltése és a függelék ide tartozó részének átolvasása után, könnyen elvégezheti az olvasó.

1. fejezet

A vizsgált polinom-differenciálegyenlet

A szakdolgozatban az alábbi komplex polinom-differenciálegyenletet vizsgálom.

$$\begin{aligned} &p, q : \mathbb{C} \to \mathbb{C}, \quad K \in \mathbb{C} \setminus \{0\}, \quad n \in \mathbb{N}, \quad n \geq 2 \\ &p(z) = z^n + a_{n-1}z^{n-1} + ... + a_0 \\ &q(z) = z^{n+1} + b_n z^n + ... + b_0 \\ &(n+1)p'q - npq' = K \end{aligned} \tag{1.1}$$

A fenti polinom-differenciálegyenlet egy speciális változata a következőnek.

$$\begin{aligned} &k, l \in \mathbb{C} \quad c \in \mathbb{C} \setminus \{0\} \quad m \in \mathbb{N} \\ &kp'q - lpq' = c \cdot p^m \end{aligned} \tag{1.2}$$

ahol p, q tetszőleges komplex együtthatós polinomokat jelölnek. Az 1.2-es polinom-differenciálegyenletet a komplex számok testére kimondott Jacobi-sejtés motiválja. Az [1] forrás jelöléseit, és elnevezéseit használva ha f, g olyan kétváltozós komplex polinomok, melyekre teljesül a $J(f, g) \equiv 1$ tulajdonság, akkor találhatunk $H(x, y) = x^l \cdot p(z)$ és $G(x, y) = x^k \cdot q(z)$ alakú w-homogén polinomokat melyekre $f_w^+ = (H)^r$ és $J(H, G) = c \cdot (H)^m$, ahol p,q kielégítik az 1.2-es polinom differenciálegyenletet.

A [1]-ben kombinatorikus leírás történt az 1.2-es polinom differenciálegyenlet megoldásaira, ami azonban nem adja meg a numerikus megoldásokat. Célom, az 1.1 polinom-differenciálegyenlet megoldásainak numerikus meghatározása $n = \{2, 3, 4\}, K = 1$ esetben.

1.1. Általános megoldások

Ebben a fejezetben pár állításban bebizonyítom, hogy egyes speciális alakú n-től függű p, q polinomok általános megoldásai lesznek a vizsgált 1.1 polinom-differenciálegyenletnek.

1.1.1. Triviális megoldás

1.1.1. Állítás. *Tetszőleges $n \geq 2$-re létezik olyan $\alpha, \beta \in \mathbb{C}$, melyekre $p(z) = z^n + \alpha$ és $q(z) = z^{n+1} + \beta z$ megoldásai az 1.1-nek.*

Bizonyítás. A vizsgált differenciálegyenletbe behelyettesítve a következőt kapjuk:

$$(n+1)n \cdot z^{n-1} \cdot (z^{n+1} + \beta z) - n \cdot (z^n + \alpha) \cdot \Big((n+1)z^n + \beta\Big) = K$$

$$(n+1)n \cdot (z^{2n} + \beta z^n) - n \cdot \Big[(n+1)z^{2n} + \Big(\alpha(n+1) + \beta\Big) \cdot z^n + \alpha\beta\Big] = K$$

Látható, hogy a $n \geq 2$ feltétel miatt a z^{2n}-hez tartozó együtthatók kiejtik egymást, így az egyenletünk a következő alakra egyszerűsödik:

$$\Big[(n+1)\beta - \alpha(n+1) - \beta\Big] \cdot z^n - \alpha\beta = \frac{K}{n}$$

Mivel K konstans, ezért az α, β komplexeknek az alábbi egyenletrendszert kell kielégíteniük:

$$\begin{aligned} n\beta - \alpha(n+1) &= 0 \\ -n\alpha\beta &= K \end{aligned}$$

Az egyenletrendszerből $\alpha = \frac{n\beta}{n+1}$ kifejezhető. Ezt visszahelyettesítve a következőt kapjuk:

$$\beta = \frac{\sqrt{-K(n+1)}}{n}$$

Tehát beláttuk, hogy $n \geq 2$-re tudunk alkalmas α, β együtthatókat választani. Speciálisan $n = 1$ esetén az összevonás után az alábbi egyenletet kapjuk.

$$(2\alpha - 2) \cdot z^2 + (2\alpha\beta - 2\alpha - \beta) \cdot z - \alpha\beta = K$$

Látszik, hogy csak az $\alpha := 1$, $\beta := 2$ lesz jó választás, viszont ekkor csak rögzített $K = -2$ esetén lesz ez megoldása a polinom-differenciálegyenletnek. □

1.1.2. Példa. A fenti állítást, és a levezetett formulát felhasználva $n = 2$ esetben $p(z) = z^2 + \frac{\sqrt{3}}{3}i$ és $q(z) = z^3 + \frac{\sqrt{3}}{2}i \cdot z$ megoldásai 1.1-nek.

1.1.3. Megjegyzés. Érdemes megjegyezni, hogy $K > 0$ esetén $\Im(\alpha), \Im(\beta) \neq 0$, és $K < 0$ esetén $\Im(\alpha), \Im(\beta) = 0$. Tehát a 3. fejezetben a kapott triviális megoldásoknak a $K = 1$ választás miatt mindig képzetesek lesznek.

1.1.2. Csebisev-polinomok transzformáltjai

A Csebisev-polinomok a numerikus analízis, egyik fontos eszköze. Felhasználásuk rendkívül széleskörű, de különösen a numerikus integrálás és a polinom interpoláció során

bizonyulnak hatékonynak [4]. Hasznos tulajdonságain túl, mint látni fogjuk, a vizsgált polinom-differenciálegyenlet egyik általános megoldását is meghatározhatjuk belőlük.

Az ortogonális polinomokat az $L_{2,\alpha}([a,b])$ téren szokás definiálni, ahol $\alpha : [a,b] \to \mathbb{R}$, $\alpha(x) \geq 0\ \forall x \in [a,b]$ és $\int_a^b \alpha(x)dx > 0$. Ezen a téren a p_n polinomrendszer α-ortogonális, ha $< p_i, p_j >_\alpha := \int_a^b p_i(x)p_j(x)\alpha(x)dx = 0\ \forall i \neq j$. A T_n elsőfajú Csebisev-polinomok, olyan speciális ortogonális polinomok, ahol $\alpha(x) := \frac{1}{\sqrt{1-x^2}}$, [a,b]=[-1,1], továbbá feltesszük, hogy $T_0 \equiv 1$. Ezekből levezethető, hogy az elsőfajú Csebisev-polinomok sorozatára az alábbi rekurzív formula írható fel.

$$\begin{aligned} T_0(x) &:= 1 \\ T_1(x) &:= x \\ T_n(x) &:= 2xT_{n-1}(x) - T_{n-2}(x) \quad n \geq 2 \end{aligned} \tag{1.3}$$

A legtöbb esetben a fenti rekurzív formulával definiálják az elsőfajú Csebisev-polinomokat. Azonban a dolgozatban az alábbi trigonometrikus alakot tekintem az elsőfajú Csebisev-polinomok definíciójának.

1.1.4. Definíció. ((elsőfajú) Csebisev-polinom) $T_n(z) := \cos\left(n \cdot \arccos(z)\right)$

1.1.5. Állítás. *$T_n(z) \in \mathbb{C}[z]$, azaz $T_n(z)$ valóban egy n-edfokú polinomot jelöl.*

Bizonyítás. Első lépésként megpróbáljuk felírni a arccos függvényt a négyzetgyök, és log függvényekkel.

$$w = \arccos(z) \Leftrightarrow \cos(w) = z \Leftrightarrow \frac{e^{iw} + e^{-iw}}{2} = z$$

Ezután az $a := e^{iw}$ választással az $a^2 + 1 - 2za = 0$ másodfokú egyenletet kapjuk, melynek megoldásai a komplex gyökvonás többértékűsége miatt az $a_{1,2} = z + \sqrt{z^2 - 1}$. Az a kifejezés megválasztásából átrendezéssel adódik a következő

$$\arccos(z) = -i \cdot \log(z + \sqrt{z^2 - 1}) \tag{1.4}$$

Ahhoz, hogy ezt a formulát be tudjuk helyettesíteni a definícióba be kell látnunk, hogy $T_n(z)$ valóban függvény lesz.

Tegyük fel, hogy $\alpha \neq \beta$-ra $\cos\alpha = \cos\beta$. Ekkor az $A := e^{i\alpha}, B := e^{i\beta}$ helyettesítésből és az Euler-képletből adódik

$$A + \frac{1}{A} = B + \frac{1}{B}$$

Ennek $e^{i\alpha} = e^{i\beta}$, és $e^{i\alpha} = e^{-i\beta}$ a megoldásai. Ekkor e^z periodikusságát felhasználva kapjuk, hogy $\beta = \alpha + 2k\pi$, illetve $\beta = -\alpha + 2k\pi$, ahol $k \in \mathbb{Z}$. De a cos párossága miatt ezek nem számítanak különböző megoldásnak, így $\cos(n \arccos(z))$ valóban függvény. Most már behelyettesíthetjük 1.4-et.

$$\cos(n\arccos(z)) = \frac{e^{n\log(z+\sqrt{z^2-1})} + e^{-n\log(z+\sqrt{z^2-1})}}{2} = \frac{1}{2}\cdot((z+\sqrt{z^2-1})^n + (z+\sqrt{z^2-1})^{-n})$$

Ezután egy gyöktelenítés, és a nevezetes azonosságok felhasználásával kapjuk a következőt.

$$\cos(n\arccos(z)) = \frac{1}{2}\cdot((z+\sqrt{z^2-1})^n + (z-\sqrt{z^2-1})^n) = T_n(z)$$

Ebből már látszik, hogy $T_n(z)$ egy pontosan n-edfokú polinom, hiszen a legnagyobb nem nulla együttható, hoz tartozó tag fokszáma n, és a négyzetgyökös tagok kiesnek. □

1.1.6. Állítás. (valós eset) *A $p(x) = T_n(x)$ és $q(x) = T_{n+1}(x)$ Csebisev-polinomok megoldásai az $(n+1)\cdot p'\cdot q - n\cdot p\cdot q' \equiv -(n+1)\cdot n$ polinom-differenciálegyenletnek.*

Bizonyítás. Az alábbi polinom-differenciálegyenletre látjuk be, hogy az állításbeli Csebisev-polinomok megoldások lesznek:

$$(n+1)\cdot p'\cdot q - n\cdot p\cdot q' \equiv M \in \mathbb{R} \tag{1.5}$$

A bizonyítás során szükségünk lesz a Csebisev-polinomok deriváltjaira.

$$T_n'(x) = -\sin\big(n\cdot\arccos(x)\big)\cdot\frac{-n}{\sqrt{1-x^2}}$$

$$T_{n+1}'(x) = -\sin\big((n+1)\cdot\arccos(x)\big)\cdot -\frac{n+1}{\sqrt{1-x^2}}$$

A helyettesítést elvégezve az alábbi egyenletet kapjuk.

$$\frac{n(n+1)}{\sqrt{1-x^2}}\cdot\Big[\sin\big(n\cdot\arccos(x)\big)\cdot\cos\big((n+1)\cdot\arccos(x)\big) - sin\big((n+1)\cdot\arccos(x)\big)\cdot\cos\big(n\cdot\arccos(x)\big)\Big] \equiv M$$

A $\sin(x-y) = \sin(x)\cos(y) - \cos(x)\sin(y)$ addíciós képletet felhasználva a következő kifejezésre egyszerűsödik a korábbi egyenlet.

$$\frac{n(n+1)}{\sqrt{1-x^2}}\cdot\sin\big(n\cdot\arccos(x) - (n+1)\cdot\arccos(x)\big) \equiv M$$

Az összevonások elvégzése után ezt kapjuk.

$$\frac{n(n+1)}{\sqrt{1-x^2}}\cdot\sin\big(-\arccos(x)\big) \equiv M$$

$$-n(n+1)\cdot\Big(\cos\big(\arccos(x)\big)\Big)' \equiv M$$

Az arccos definícióját felhasználva valóban adódik, hogy $M = -(n+1)\cdot n$. Tehát a $T_n(x)$ és $T_{n+1}(x)$ polinomok valóban megoldásai az állításban szereplő polinom-

differenciálegyenletnek. □

1.1.7. Állítás. (komplex eset) *A $p(z) = T_n(z)$ és $q(z) = T_{n+1}(z)$ Csebisev-polinomok megoldásai az $(n+1) \cdot p' \cdot q - n \cdot p \cdot q' \equiv -(n+1) \cdot n$ polinom-differenciálegyenletnek.*

Bizonyítás. Az 1.1.5-os állításból következik, hogy $2n$-edfokú komplex polinom áll az átrendezett polinom-differenciálegyenlet bal oldalán.

$$(n+1) \cdot T_n'(z) \cdot T_{n+1}(z) - n \cdot T_n(z) \cdot T_{n+1}'(z) + (n+1) \cdot n = 0$$

Tehát elég lenne belátni, hogy a bal oldalon álló polinom legalább $2n+1$ helyen 0. A valós esetből pedig következik, hogy tetszőleges $[-1, 1]$-beli valós számra a fenti egyenlet bal oldala 0, ezért a bal oldal valóban az azonosan 0 polinom, azaz $p(z) = T_n(z)$ és $q(z) = T_{n+1}(z)$ valóban megoldásai az állításban szereplő polinom-differenciálegyenletnek. □

A célom most az, hogy az előző állításokat felhasználva a $T_n(z)$ és $T_{n+1}(z)$ Csebisev-polinomokat úgy transzformáljam, hogy azok az eredeti 1.1-es polinom-differenciálegyenletnek a normált n illetve $n+1$-edfokú megoldásai legyenek.

1.1.8. Állítás. *Tetszőleges $n \geq 2$-re $p(z) = c_1 \cdot T_n(bz)$ és $q(z) = c_2 \cdot T_{n+1}(bz)$ megoldásai az 1.1 polinom-differenciálegyenletnek. Ahol $c_1 = \frac{1}{2^{n-1}b^n}$, $c_2 = \frac{1}{2^n b^{n+1}}$ és $b = \sqrt[2n]{\frac{-n \cdot (n+1)}{2^{2n-1} \cdot K}}$ alkalmas konstansok.*

Bizonyítás. A Csebisev-polinomok rekurzív képletéből (1.3) indukcióval belátható, hogy $n \geq 2$-re a $T_n(z)$ főegyüttható 2^{n-1}. Tehát $T_n(bz)$ főegyüttható $2^{n-1} \cdot b^n$, és $T_{n+1}(bz)$ főegyüttható $2^n \cdot b^{n+1}$. Ebből következik, hogy a $c_1 = \frac{1}{2^{n-1}b^n}$, $c_2 = \frac{1}{2^n b^{n+1}}$ konstansokkal p és q normált polinomok.

Most használjuk fel az 1.1.7-es állítást, miszerint

$$(n+1) \cdot T_n(z)' \cdot T_{n+1}(z) - n \cdot T_n(z) \cdot T_n + 1(z)' \equiv -(n+1) \cdot n$$

Ebből következik, hogy

$$c_1 c_2 \cdot \Big((n+1) \cdot T_n(bz)' \cdot T_{n+1}(bz) - n \cdot T_n(bz) \cdot T_{n+1}(bz)' \Big) \equiv -c_1 c_2 \cdot (n+1) \cdot n \cdot b$$

Tehát az állításban szereplő p és q megoldásai lesznek az 1.1-nek, ha teljesül az alábbi egyenlet.

$$K = -c_1 c_2 \cdot (n+1) \cdot n \cdot b = \frac{-(n+1) \cdot n}{2^{2n-1} \cdot b^{2n}} \tag{1.6}$$

Ebből már adódik, hogy $b = \sqrt[2n]{\frac{-n \cdot (n+1)}{2^{2n-1} \cdot K}}$. □

1.1.9. Példa. $n = 3$ és $K = 1$ esetén próbáljuk meghatározni a megfelelő Csebisev-polinomokból transzformált megoldásokat.

Egy *Maple* programcsomagban készített *csebisev_mo_tester* függvénnyel az 1.1.8 állításban meghatározott konstansok képleteit felhasználva a Csebisev-polinomokból transzformált megoldások a következők.

A függvény hívása: *csebisev_mo_tester(3,1);*

"Csebisev-polinomok:"

$$4\,z^3 - 3\,z$$

$$1 + 8\,z^4 - 8\,z^2$$

"Transzformált Csebisev-polinomok:"

$$p = z^3 + (-0.5200209565 + 0.9007027170\,I)\,z$$

$$q = z^4 + (-0.6933612751 + 1.200936956\,I)\,z^2 - 0.1201874642 - 0.2081707948\,I$$

$$p = z^3 + 1.040041912\,z$$

$$q = z^4 + 1.386722549\,z^2 + 0.2403749284$$

$$p = z^3 + (-0.5200209565 - 0.9007027170\,I)\,z$$

$$q = z^4 + (-0.6933612751 - 1.200936956\,I)\,z^2 - 0.1201874642 + 0.2081707948\,I$$

1.1. ábra. csebisev_mo_tester output: n=3,K=1

Láthatjuk, hogy az egyik megoldás csupa valós együtthatós polinomokból áll, erre a konkrét megoldásra a későbbiekben még hivatkozni fogok.

$$\begin{aligned} p(z) &= z^3 + 1.04 \cdot z \\ q(z) &= z^4 + 1.39 \cdot z^2 + 0.24 \end{aligned} \tag{1.7}$$

1.2. Megoldások vizsgálata közönséges differenciálegyenlettel

Az 1.1 probléma vizsgálatára az egyik lehetőség, ha elsőrendű lineáris közönséges differenciálegyenletre vezetjük vissza. Ennek célja nem a megoldások meghatározása, hanem egy alkalmazásuk bemutatása.

Tegyük fel, hogy ismerjük a q sehol sem 0 polinomot. Ekkor átrendezhetjük a vizsgált polinom-differenciálegyenletet a következő módon.

$$p'(z) - \frac{n}{n+1} \cdot \frac{q'(z)}{q(z)} p(z) = \frac{K}{n+1} \cdot \frac{1}{q(z)}$$

$$p'(z) = \frac{n}{n+1} \cdot \frac{q'(z)}{q(z)} p(z) + \frac{K}{n+1} \cdot \frac{1}{q(z)} = a(z) \cdot p(z) + b(z) \tag{1.8}$$

Tegyük fel, hogy $p_0(z)$ megoldása a homogén $K = 0$ esetnek, azaz $p_0'(z) = a(z) \cdot p_0(z)$. Ekkor a lineáris differenciálegyenlet megoldásai $p(z) = W(z) \cdot p_0(z)$ alakúak, ahol W differenciálható és $p_0(z)$ sem veszi fel a 0-t, illetve teljesül

$$W'(z) = \frac{b(z)}{p_0(z)} \tag{1.9}$$

Használjuk fel, hogy $p_0(z)$ a homogén eset megoldása.

$$p'(z) = W'(z) \cdot p_0(z) + W(z) \cdot p_0'(z) = W'(z) \cdot p_0(z) + W(z) \cdot a(z) \cdot p_0(z) = b(z) + a(z) \cdot p(z)$$

A homogén esetben könnyen találhatunk $p_0(z)$ megoldást. Ekkor ugyanis 1.8 a következő alakba írható

$$\frac{p_0'(z)}{p_0(z)} = \frac{n}{n+1} \cdot \frac{q'(z)}{q(z)}$$

Mindkét oldalt integrálva kapjuk

$$\log p_0(z) = \frac{n}{n+1} \cdot \log q(z) + C_1$$

Tehát például $C_1 := 0$ választással $p_0 = q(z)^{\frac{n}{n+1}}$ megoldása lesz a homogénnek. Ezt helyettesítsük a 1.9 egyenletbe, továbbá használjuk fel az 1.8-ban a b-re kapott formulát.

$$W'(z) = \frac{K}{n+1} \cdot \frac{1}{q(z)} \cdot q(z)^{-\frac{n}{n+1}} = \frac{K}{n+1} \cdot q(z)^{-\frac{2n+1}{n+1}}$$

Végül kaptuk, hogy W a következő képpen áll elő

$$W(z) = \frac{K}{n+1} \cdot \int q(z)^{-\frac{2n+1}{n+1}} dz \tag{1.10}$$

Továbbá ha p valóban egy n-edfokú polinom, ami megoldása az eredeti lineáris differenciálegyenletnek akkor W az alábbi alakba írható.

$$W(z) = p(z) \cdot q(z)^{-\frac{n}{n+1}} \tag{1.11}$$

Hiszen ekkor $p(z) = p(z) \cdot q(z)^{-\frac{n}{n+1}} \cdot q(z)^{\frac{n}{n+1}}$ valóban teljesül.

A fent kapott eredményeket használjuk fel az $\int q(z)^{-\frac{2n+1}{n+1}}$ alakú integrálok kiszámítására. Ha p és q az 1.1 megoldásai, akkor a fenti levezetés 1.11 és 1.10 egyenleteiből kapjuk, hogy

$$\int q(z)^{-\frac{2n+1}{n+1}} dz = p(z) \cdot q(z)^{-\frac{n}{n+1}} \cdot \frac{n+1}{K} + C_2$$

Ez az eredmény azért jelentős, mert általában egy több monomból álló polinom racionális hatványának ritkán van elemi primitív függvénye.

1.2.1. Példa. Az 1.1.9-es példában beláttam, hogy az 1.7-ben szereplő polinomok a vizsgált polinom-differenciálegyenlet megoldásainak kerekítései az $n = 3$, $K = 1$ esetben. Tehát a kerekítésből adódó hibától eltekintve teljesülni fog a következő

$$\int \sqrt[4]{(z^4 + 1.39 \cdot z^2 + 0.24)^{-7}} dz = 4 \cdot (z^3 + 1.04 \cdot z) \cdot \sqrt[4]{(z^4 + 1.39 \cdot z^2 + 0.24)^{-3}} + C$$

Tehát a példában is látható, hogy a p,q-ra vonatkozó ismeretek jelentősen megkönnyítették a primitív függvény megtalálását.

1.3. Megoldások vizsgálata egyenletrendszerrel

Az 1.1 probléma megoldását megkaphatjuk, ha egy nemlineáris egyenletrendszerre vezetjük vissza a polinom-differenciálegyenletet, ahol a változók a polinomok együtthatói lesznek.

$$\begin{aligned} &p, q : \mathbb{C} \to \mathbb{C}, K \in \mathbb{C} \setminus \{0\} \\ &p(z) = z^n + a_{n-1}z^{n-1} + a_{n-2}z^{n-2} + ... + a_0 \\ &q(z) = z^{n+1} + b_n z^n + b_{n-1}z^{n-1} + ... + b_0 \\ &(n+1)p'q - npq' = K \end{aligned} \tag{1.12}$$

Helyettesítsük a $p(z)$ és $q(z)$ polinomokat a polinom-differenciálegyenletbe. Ekkor könnyen láthatjuk, hogy a z^{2n} tagok kiejtik egymást, ha $n \geq 2$. Tehát a következő egyenletnek kell teljesülnie, ahol f_j $j = 0...(2n-1)$ függvények mind többváltozós polinomok.

$$f_{2n-1}(a_0, ..., a_{n-1}, b_0, ..., b_n) \cdot z^{2n-1} + ... + f_0(a_0, ..., a_{n-1}, b_0, ..., b_n) = K$$

A fenti egyenlet teljesül $\forall\ z \in \mathbb{C}$-re, azaz p,q megoldásai lesznek a polinomdifferenciálegyenletnek, ha teljesül a következő egyenletrendszer

$$\begin{aligned} f_{2n-1}(a_0, ..., a_{n-1}, b_0, ..., b_n) &= 0 \\ f_{2n-2}(a_0, ..., a_{n-1}, b_0, ..., b_n) &= 0 \\ &\vdots \\ f_1(a_0, ..., a_{n-1}, b_0, ..., b_n) &= 0 \\ f_0(a_0, ..., a_{n-1}, b_0, ..., b_n) &= K \end{aligned}$$

Azonban az egyenletrendszer megoldásait még nem tudnánk egyértelműen meghatározni, hiszen csak $2n$ egyenletünk, és $2n+1$ változónk van, ezért vizsgáljuk meg, hogy vannak-e olyan változók melyeket eliminálhatunk.

Azt biztosan állíthatjuk, hogy $w := z + \frac{a_{n-1}}{n}$ helyettesítéssel eliminálhatjuk p-ből a w^{2n-1} tagot. Ekkor tehát 1.12 az alábbi alakra hozható

$$\begin{aligned} &p, q : \mathbb{C} \to \mathbb{C}, K \in \mathbb{C} \\ &p(z) = w^n + \tilde{a}_{n-2}w^{n-2} + ... + \tilde{a}_0 \\ &q(z) = w^{n+1} + \tilde{b}_n w^n + \tilde{b}_{n-1}w^{n-1} + ... + \tilde{b}_0 \\ &(n+1)p'q - npq' = K \neq 0 \end{aligned} \tag{1.13}$$

Ekkor belátható, hogy $\tilde{b}_n = 0$ is teljesül. Vizsgáljuk az $f_{2n-1}(\tilde{a}_0, ..., \tilde{a}_{n-2}, \tilde{b}_0, ..., \tilde{b}_n) = 0$ egyenletet. Használjuk ki, hogy w^{2n-1} együtthatója a megfelelő tagok összeszorzása után az alábbi lesz

$$(n+1)n \cdot \tilde{b}_n - n^2 \cdot \tilde{b}_n = n \cdot \tilde{b}_n = 0$$

Tehát kaptuk, hogy $\tilde{b}_n = 0$, ha q megoldás. Így a változóink száma $2n-1$, illetve pontosan $2n-1$ egyenletet nem használtunk még fel, tehát az egyenletrendszer egy $f : \mathbb{C}^{2n-1} \to \mathbb{C}^{2n-1}$ leképezésnek felel meg melynek minden koordinátafüggvénye többváltozós polinom. A továbbiakban mindig felteszem, hogy 1.12 jelöléseit használva $a_{n-1} = b_n = 0$.

Az eljárást a következő példán keresztül is szemléltetem.

1.3.1. Példa. Legyen $n := 3$ és $K := 1$. Ekkor a probléma a következőképpen néz ki:

$$\begin{aligned} &p(z) = z^3 + az^2 + bz + c \\ &q(z) = z^4 + dz^3 + ez^2 + fz + g \\ &4p'q - 3pq' = 1 \end{aligned}$$

A változók helyettesítésével feltehető, hogy $a = d = 0$. A polinom-differenciálegyenletbe helyettesítve ezt kapjuk:

$$4(3z^2 + b)(z^4 + ez^2 + fz + g) - 3(z^3 + bz + c)(4z^3 + 2ez + f) = 1$$

$$\begin{aligned} &\Big(12z^6 + (4b + 12e) \cdot z^4 + 12fz^3 + (4be + 12g) \cdot z^2 + 4bfz + 4bg\Big) - \\ &\Big(12z^6 + (12b + 6e) \cdot z^4 + (12c + 3f) \cdot z^3 + 6bez^2 + (6ce + 3bf) \cdot z + 3cf\Big) = 1 \end{aligned}$$

Az összevonások elvégzése után az egyenlet a következő:

$$(-8b + 6e) \cdot z^4 + (-12c + 9f) \cdot z^3 + (-2be + 12g) \cdot z^2 + (bf - 6ce) \cdot z + 4bg - 3cf = 1$$

Ebből az alakból láthatjuk, hogy p,q megoldása lesz a polinom-differenciálegyenletnek, ha az együtthatóik egy megoldását adják az alábbi nemlineáris egyenletrendszernek.

$$\begin{aligned} -8b + 6e &= 0 \\ -12c + 9f &= 0 \\ -2be + 12g &= 0 \\ bf - 6ce &= 0 \\ 4bg - 3cf &= 1 \end{aligned} \tag{1.14}$$

Az egyenletrendszert a *Maple* programcsomag beépített *solve* parancsával megoldva a következő megoldásokat kapjuk.

```
evalf(solve({-8*b+6*e,-12*c+9*f,-2*b*e+12*g,b*f-6*c*e,4*b*g-3*c*f-1},{b,c,e,f,g}));
{b = 1.040041912, c = 0., e = 1.386722549, f = 0., g = 0.2403749284}, {b = 0., c = 0.5000000000 I, e = 0., f = 0.6666666667 I, g = 0.}
```

1.2. ábra. Maple:n=3,K=1

Ekkor jelöljük a két megoldást a következőképpen.

$$\begin{aligned} p_1(z) &= z^3 + \frac{1}{2}I \\ q_1(z) &= z^4 + \frac{2}{3}I \cdot z \end{aligned}$$

$$\begin{aligned} p_2(z) &= z^3 + 1.040041912 \cdot z \\ q_2(z) &= z^4 + 1.386722549 \cdot z^2 + 0.2403749284 \end{aligned}$$

Ekkor látható, hogy p_1 és q_1 épp a triviális megoldásnak felel meg, ha felhasználjuk a 1.1.1-es állítás bizonyításában kapott együttható formulákat.

$$\begin{aligned} f &= \frac{\sqrt{-(n+1)}}{n} = \frac{\sqrt{-4}}{3} = \frac{2}{3}I \\ c &= \frac{\sqrt{-(n+1)}}{n+1} = \frac{\sqrt{-4}}{4} = \frac{1}{2}I \end{aligned}$$

Illetve p_2 és q_2 megegyeznek az 1.7-ban szereplő valós együtthatós Csebisev-polinomokból transzformált megoldásokkal.

Tehát a fenti példában látható módon tetszőleges n-re a polinom-differenciálegyenlet visszavezethető egy nemlineáris egyenletrendszerre, melynek megoldására számos módszer létezik. Ezek közül az egyik legjelentősebb a Newton–Raphson-módszer, illetve speciális változatai. Ezek taglalásával foglalkozik a következő fejezet.

2. fejezet

Newton–Rapshon-módszer

A numerikus matematikában egyenletrendszerek megoldására leggyakrabban a Newton–Raphson-módszert használják. Ez főleg annak köszönhető, hogy könnyen programozható iterációs lépéseket használ. Sok helyen csak Newton-módszerként említik.

Tegyük fel, hogy adott $f : \mathbb{R} \to \mathbb{R}$ folytonosan differenciálható függvény, és $f(x^*) = 0$. Ekkor bizonyos feltételeket kielégítő f-re és az $x_{m+1} := x_m - \frac{f(x_m)}{f'(x_m)}$ sorozatra $x_m \to x^*$, ha x_0 elég közel van x^*-hoz. Ebben az egy dimenziós esetben még jól látható a módszer geometriai jelentése. A képlet az l_m-mel jelölt x_m-beli érintő egyenletéből egy átrendezéssel következik, ahol x_{m+1}-et az x_m-beli érintő és az abcissza metszéspontjának választottuk.

$$l_m(x_{m+1}) = f(x_m) + f'(x_m)(x_{m+1} - x_m) = 0 \tag{2.1}$$

A Newton–Rapshon-módszer széleskörű felhasználásának egyik oka, hogy általánosítható többváltozós valós értékű függvények, sőt Banach-terek közötti leképezések gyökeinek keresésére is [5]. A módszer egyik további fontos tulajdonsága, hogy a gyökhöz elég közelről indítva a konvergenciasebesség négyzetes (kvadratikus). Az alábbi két példában a Newton–Raphson-módszer egy egy alkalmazását mutatom be.

2.0.2. Példa. (Heron-algoritmus) Adott $a \in \mathbb{R}$ esetén $x_n \to \sqrt{a}$, ahol $x_{n+1} := \frac{1}{2} \cdot \left(x_n + \frac{a}{x_n}\right)$. Itt a Newton-módszert az $f(x) = x^2 - a$ valós függvényre alkalmaztuk. [2][381.o.]

2.0.3. Példa. (Schultz–Hotelling–Bodewig-algoritmus) Tekinsük az $F : \mathbb{M}_{N\times N} \to \mathbb{M}_{N\times N}, F(X) := X^{-1} - A$ leképezésre alkalmazzuk a Newton-módszert. Kapjuk, hogy az $X_{n+1} := X_n(2 \cdot I_N - AX_n)$ iteráció esetén $X_n \to A^{-1}$. A numerikus analízis 2. tantárgy előadásán csak Schultz-módszerként említettük ezt az algoritmust.

2.1. Konvergencia tételek

Az 1.3 részben levezettem, hogy a vizsgált polinom-differenciálegyenlet visszavezethető, egy nemlineáris egyenletrendszer megoldására, melyet az általam készített programban

többdimenziós Newton–Raphson-módszerrel valósítottam meg. A tételeket is több dimenzióban fogom kimondani.

2.1.1. Jelölés. Adott $x \in \mathbb{R}^n$ $\epsilon > 0$ sugarú környezetét jelölje $B_\epsilon(x)$.

2.1.2. Tétel. (Lokális konvergencia) *Adott $H \subseteq \mathbb{R}^n$, $f : H \to \mathbb{R}^n$ és létezik $x^* \in H$, melyre $f(x^*) = 0$. Ekkor ha f' Lipschitz-folytonos H-ban L Lipschitz-állandóval, továbbá létezik $\alpha > 0$, melyre $\|f'(x)^{-1}\| \leq \alpha$ minden $x \in H$-ra, akkor $x_{m+1} = x_m - f'(x_m)^{-1} \cdot f(x_m)$ esetén teljesül:*

1. *van olyan $\delta > 0$, melyre tetszőleges $x_0 \in B_\delta(x^*)$-re $\lim_{m\to\infty} x_m = x^*$.*
2. $\|x_{m+1} - x_*\| \leq C\|x_m - x^*\|^2 \quad C := \frac{L\alpha}{2}$

A bizonyítást csak $n = 1$ esetén közlöm [2][372.o.]. A [2][382.o]-on szerepel, hogy ez a bizonyítás általánosítható több dimenzióra.

Bizonyítás. Írjuk át az iterációra adott formulát a következőképpen.

$$x_{m+1} - x^* = x_m - x^* - (f'(x_m))^{-1}\Big(f(x_m) - f(x^*)\Big) = -(f'(x_m))^{-1}\Big(f(x_m) - f(x^*) - f'(x_m)(x_m - x^*)\Big) \tag{2.2}$$

Definiáljuk az r segédfüggvényt az alábbi módon, illetve használjuk a Newton–Leibniz szabályt a koordinátánkénti integrálásnál.

$$r(x, y) = f(x) - f(y) - f'(x)(x - y) = \int_y^x \Big(f'(z) - f'(x)\Big)dz \tag{2.3}$$

Helyettesítsünk $z = y + t(x - y)$-t az integrálba és használjuk fel f' Lipschitz-folytonosságát.

$$|r(x, y)| \leq |x - y| \int_0^1 |f'(y + t(x - y)) - f'(x)|dt \leq L|x - y|^2 \int_0^1 (1 - t)dt = \frac{L}{2}|x - y|^2 \tag{2.4}$$

Vegyük észre, hogy a 2.2-es egyenlet 2. tényezője épp $r(f(x_m), f(x^*)$. Használjuk fel a Jacobi-mátrix inverzének korlátosságát.

$$|x_{m+1} - x^*| \leq \alpha \cdot \frac{L}{2} \cdot |x_m - x^*|^2$$

Ezzel a tétel 2. állítását beláttuk.

Most már csak a konvergenciát kell belátni. A módszer beindításakor teljesüljön a következő:

$$C|x_0 - x_*| \leq q < 1 \tag{2.5}$$

ekkor az előző egyenlőtlenségből ,és a 2. állításból következik, hogy:

$$|x_1 - x^*| \leq q|x_0 - x^*|$$

és hasonlóan teljesülni fog

$$|x_m - x^*| \leq q|x_{m-1} - x^*| \quad m \geq 1 \tag{2.6}$$

Így az $\epsilon_m := |x_m - x^*|$ egy konvergens mértani sor lesz, amiből következik, hogy

$$\lim_{m\to\infty} x_m = x^*$$

Ekkor az 2.5 egyenlőtlenséget felhasználva a $\delta < \frac{1}{C}$ választás megfelelő lesz. □

2.1.3. Tétel. (Monoton konvergencia) *Adott $H \subseteq \mathbb{R}^n$, $f : H \to \mathbb{R}^n$, $f \in C^2(H)$ és $f'(x) \neq 0$, illetve $f''(x) \neq 0$ tetszőleges $x \in H$-ra. Továbbá létezik $x^* \in H$, melyre $f(x^*) = 0$. Ekkor $x_{m+1} = x_m - f'(x_m)^{-1} \cdot f(x_m)$ esetén, tetszőleges $x_0 \in H$-ra az $x_m \to x^*$ monoton konvegencia teljesül, ha $f(x) > l_0(x) = f(x_0) + f'(x_0)(x - x_0)$ és $f(x_0) > 0$. [2][380-382.o.]*

2.1.4. Megjegyzés. A fenti tételben $f'(x) \neq 0$ kikötés azért fontos, mert a Newton-módszer jelentősen lelassul olyan gyök közelében, ami lokális szélsőérték is egyben. Ezt a módszer mértani jelentése is szemléletesen alátámasztja az egydimenziós esetben. Tehát a Newton–Raphson-módszer gyakorlati megvalósításakor rögzítenünk kell egy maximális iterációszámot, illetve érdemes bevezetni egy $\epsilon_\Delta > 0$ viszonylag kicsi számot, amire $\|f(x_m) - f(x_{m-1})\| < \epsilon_\Delta$ feltétel teljesülésekor szintén megállítjuk az algoritmust azt feltételezve, hogy $f'(x^*) = 0$.

2.1.5. Megjegyzés. A fenti tételben az $f(x) > l_0(x)$ egyenlőtlenséget komponensenként értjük. Egy dimenzióban könnyen látható, hogy globálisan konvex függvény és $f(x_0) > 0$ esetén az x_0 kezdeti közelítés x^*-hoz viszonyított helyzetétől függően az x_n monoton nő, vagy csökken, és így konvergál x^*-hoz. $f(x_0) < 0$-ra is teljesül a konvergencia, de csak pár lépés után lesz monoton. Az igazán fontos feltétel az $f''(x) \neq 0$. Ha f-nek lenne inflexiós pontja, akkor az iteráció akár végtelen ciklusba is futhat. Egydimenziós esetben például $f(x) = x^3 - 2x + 2$ az $x_0 := 0$ választással [5].

Magasabb dimenzióban fontos a módszer globalizálása, hiszen a megoldásokat körülvevő vonzási gömbök átmérője a dimenzió növelésével csökkennek. A globalizált módszer konvergencia sebessége lelassul a kvadratikushoz képest, de ezáltal egy a gyakorlatban is használható módszert kaphatunk, mert nincs szükség elég jó kezdeti közelítésre az iteráció beindításához. Vektormezők esetén erre egy lehetőség a *folytatás módszere* [2][388-391].

A vizsgált $0 = f(x) \in \mathbb{R}^n$ egyenletrendszert ágyazzuk, be egy $n + 1$ változós egyenletrendszerbe a következőképpen.

$$g(x, t) := f(x) + (t - 1)f_0, \quad f_0 := f(x_0) \tag{2.7}$$

Vegyük észre, hogy $t = 1$ esetén g megoldása az eredeti egyenletrendszer megoldását adja, és $t = 0$ esetben $x_0 \in \mathbb{R}^n$ megoldása g-nek. Feltehető, hogy $f_0 \neq 0$. Ekkor $t_1 :=$

$\frac{1}{k} =: \Delta t$ jelöléseket használva elég nagy k esetén x_0 jó kezdeti közelítés lesz a $g(x_1, t_1) = 0$ megoldására Newton–Raphson-módszerre, hiszen ilyenkor $\frac{1}{k} \approx 0$. Ha esetleg adott k-ra nem konvergálna a módszer a g gyökéhez, akkor csökkentsük k értékét, amíg ez be nem következik. Ezután $t_2 := t_1 + \Delta t$-re vizsgáljuk $g(x_2, t_2) = 0$ egyenletrendszert. Tehát adott $x_0 \in \mathbb{R}^n$ és $k > 1$, amit a fent leírtak alapján választottunk, ekkor $m = 0, ..., k-1$-re az

$$g(x, t_{m+1}) - g(x_m, t_m) = 0, \quad t_{m+1} := t_m + \Delta t, \quad \Delta t := 1/k \tag{2.8}$$

egyenletrendszert Newton–Raphson-módszerrel megoldva az x_{m+1}-et kaptuk. Az iterációt $t_k = 1$ értékig végezzük, hiszen az ehhez tartozó x_k a 2.7 miatt már az eredetileg vizsgált f megoldása lesz.

Tehát a folytatás módszere a Newton–Raphson-módszer többször egymás utáni végrehajtását jelenti, úgy hogy a következő köztes módszer kezdeti közelítése megegyezik az előző gyökével. A következő tétel egy elégséges feltételt ad a módszer konvergenciára.

2.1.6. Tétel. *Legyen $f(x^*) = 0$. Ekkor ha $\|f'(x)\| \leq M_1$, $\|f'(x) - f'(y)\| \leq L\|x - y\|$ illetve $\|f'(x)^{-1}\| \leq \alpha$ minden $x, y \in \mathbb{R}^n$-re. Továbbá a folytatás módszerének k lépészámára teljesül $k > L\alpha^2\|f_0\|$, akkor minden $\epsilon > 0$-hoz létezik η, a belső Newton-lépések számát jelölő konstans, melyre $\|x_k - x^*\| \leq \epsilon$. [2][389.o]*

Természetesen felmerül a kérdés, hogy vajon a Newton–Raphson-módszer mennyire használható az esetünkben, hiszen a konvergencia tételek csak az $f : \mathbb{R}^n \to \mathbb{R}^n$ vektormezőkre lettek kimondva, míg az 1.3-as fejezetben láttuk, hogy a vizsgált polinomdifferenciálegyenlet egy $f : \mathbb{C}^{2n-1} \to \mathbb{C}^{2n-1}$ leképezésre vezethető vissza, ahol n jelöli a p polinom fokszámát.

A módszert igazán akkor mondhatnánk hatékonynak, ha globális konvergenciát tapasztalnánk. A fent említett folytatás módszerének konvergenciájához vektormezők esetén az egyik elégséges feltétel volt a deriváltak globális korlátossága. Ez azonban a legalább másodfokú polinomokra nem teljesül. Ettől eltekintve egy konkrét esetre a folytatás módszere konvergálhat, ezt mutatja be a 3.0.6 példa.

Az egyváltozós komplex esetben a Newton-módszer képletéből következik, hogy az elsőrendű polinomokra teljesül a globális konvergencia. Egyváltozós másodrendű polinomok esetében egy egyenes pontjainak kivételével tetszőleges kezdeti közelítésre eljutunk a polinom valamelyik gyökéhez. Végül az ismert eredmények közé tartozik az egyváltozós harmadrendű eset is, ilyenkor pár közelítőleg nullmértékű halmaztól eltekintve a Newton–Raphson-módszer konvergál valamelyik gyökhöz [3][3.4., 3.5. ábrák], és ismert a módszernek egy olyan módosítása, melyre ezekre a kezdeti közelítésekre is konvergenssé tehető a módszer [3][4.1. megjegyzés, 4.2. ábra]. A harmadik fejezetben több dimenzióban fogom megkeresni a 1.3 fejezetben kapott $f : \mathbb{C}^{2n-1} \to \mathbb{C}^{2n-1}$ leképezés gyökeit. Ezt az magyarázza, hogy $n = 1$-re a fent említett fokszámokra a Newton-módszer nagy valószínűséggel globálisan is konvergált.

2.2. A Jacobi-mátrix közelítése

Amennyiben célunk, hogy a Newton–Raphson-módszert egy felhasználóbarát programban implementáljuk, akkor gondoskodnunk kell arról, hogy a programunk minden iterációs lépésben magától számolja ki a Jacobi-mátrix egy közelítését. Az alábbiakban erre két lehetőséget részletezek.

2.2.1. A Jacobi-mátrix közelítése definíció alapján

Tegyük, fel hogy adott $f : \mathbb{R}^n \to \mathbb{R}^n$ függvény. Ekkor jelölje $J : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ az f Jacobi-mátrixát. Definíció szerint $J = (f_{ij})_{i,j=1}^n$, ahol $f_{ij} = \frac{\partial f_i}{\partial x_j}$. Használjuk az $f'(x) = J(x)$-re az alábbi elemenkénti közelítést.

$$f_{ij} = \frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x + h_{ij}e_j) - f_i(x)}{h_{ij}} \tag{2.9}$$

ahol e_j a j-edik koordinátához tartozó egységvektort, és h_{ij} alkalmas nem nulla konstanst jelölnek. Problémát jelenthet azonban az, hogy így $f'(x)$ közelítéséhez $2 \cdot n^2$ függvényértéket kell kiszámolni, ami változók számának növelésével jelentős futásidő romlást eredményez, ezért érdemes figyelembe venni a Jacobi mátrix ritkasági struktúráját, mert így a közelítás kiszámításának műveletigénye jelentősen csökkenthető. Ez azért tehető meg, mert a nemlineáris egyenletrendszerekhez tartozó Jacobi-mátrixok általában ritka mátrixok [2][384.o].

2.2.2. A Broyden-módszer

Adott $f : \mathbb{R}^n \to \mathbb{R}^n$ gyökeinek meghatározására a Newton–Raphson-módszer az alábbi formulát használja.

$$x_{m+1} := x_m - J^{-1}(x_m) \cdot f(x_m) \quad f' = J \tag{2.10}$$

Az $n = 1$ esetben az $f'(x_m) \approx \frac{f(x_m)-f(x_{m-1})}{x_m-x_{m-1}}$ közelítést használva a módszer szuperlineárisan lokálisan konvergál, ha teljesülnek a 2.1.2 tétel feltételei. Ez a módszer a szelőmódszer.

A Broyden-módszer egy kvázi-Newton-módszer, a szelőmódszer többdimenziós általánosítása. Jelölje J_m sorozat azt a mátrixsorozatot, amire teljesül a kvázi-Newton egyenlet:

$$J_m \cdot (x_m - x_{m-1}) \approx f(x_m) - f(x_{m-1}) =: y_m \tag{2.11}$$

Továbbá J_m a J_{m-1}-től csupán egy egy rangú mátrixban különbözzön, azaz alkalmas u-ra és $s_m := x_m - x_{m-1}$-re

$$J_m - J_{m-1} = us_m^T \tag{2.12}$$

Az u vektor értéke a fenti képletből könnyen meghatározható, ha mindkét oldalt megszorozzuk s_m-mel.

$$(J_m - J_{m-1})s_m = u \cdot \|s_m\|^2 \quad \Rightarrow \quad u = \frac{(J_m - J_{m-1})s_m}{\|s_m\|^2}$$

Így az u értékét, illetve 2.11 becslést felhasználva megkapjuk a Jacobi közelítések rekurzív képletét:

$$J_m = J_{m-1} + \frac{(J_m - J_{m-1})s_m}{\|s_m\|^2} \cdot s_m^T = J_{m-1} + \frac{y_m - J_{m-1}s_m}{\|s_m\|^2} \cdot s_m^T \tag{2.13}$$

Ekkor a Broyden-módszer az x_m és x_{m-1} korábbi közelítéseket felhasználva az alábbi formula alapján számolja ki a következő közelítést

$$x_{m+1} = x_m - J_m^{-1} \cdot f(x_m) \quad m \geq 1 \tag{2.14}$$

illetve $m = 0$ esetben egy Newton-iterációt hajt végre. Ezután x_1, és x_0 közelítésekből indítható a Broyden-módszer.

2.2.1. Tétel. (Shermann–Morrison formula) *Legyen $A \in (R)^{n\times n}$ reguláris, illetve $u, v \in (R)^n$, amire $vA^{-1}u \neq -1$. Ekkor*

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u} \tag{2.15}$$

Bizonyítás. Jelölje $\alpha := \frac{1}{1+v^TA^{-1}u}$ kifejezést. Ellenőrizzük le, hogy valóban fennáll-e az inverz viszony:

$$\begin{aligned} I = (A^{-1} - \alpha \cdot A^{-1}uv^TA^{-1})(A + uv^T) = \\ = I - \alpha \cdot A^{-1}uv^T + A^{-1}uv^T - \alpha \cdot A^{-1}uv^TA^{-1}uv^T \Leftrightarrow \\ \Leftrightarrow -\alpha \cdot A^{-1}uv^T + A^{-1}uv^T - \alpha \cdot A^{-1}uv^TA^{-1}uv^T = 0 \Leftrightarrow \\ \Leftrightarrow -\alpha \cdot uv^T + uv^T - \alpha \cdot uv^TA^{-1}uv^T = 0 \Leftrightarrow \\ \Leftrightarrow \alpha \cdot (-uv^T - uv^TA^{-1}uv^T) = -uv^T \Leftrightarrow \\ \Leftrightarrow u(1 + v^TA^{-1}u) \cdot \alpha)v^T = uv^T \end{aligned}$$

A legutóbbi kifejezés pedig épp a definíciót jelenti. □

A 2.12 egyenletből átrendezéssel kapjuk, hogy $J_m = J_{m-1} + us^T$. Ha a Shermann–Morrison formula feltétele teljesül akkor kapjuk, hogy elég csak a J_m^{-1} sorozatot rekurzívan kiszámolnunk, így elég csak mátrixszorzásokat végeznünk a következő közelítés meghatározásához. A [6] forrás megemlít egy úgynevezett "Jó Broyden-módszert", ami a Jacobi-mátrix közelítéseinek inverzét az alábbi formula szerint számolja.

$$J_m^{-1} = J_{m-1}^{-1} + \frac{s - J_{m-1}^{-1}y}{s^TJ_{m-1}^{-1}y} \cdot (s^TJ_{m-1}^{-1}) \tag{2.16}$$

Ez a képlet a 2.2.1-ből levezethető. A *Matlab*-ban történő implementálásnál az előző

képletet választottam.

2.2.2. Megjegyzés. Ha teljesülnek a 2.1.2 tétel feltételei, és x_0 elég jó kezdeti közelítés, és J_0 elég jól közelíti $J(x_0)$-t, akkor a Broyden-módszer szuperlineárisan lokálisan konvergens [2][388.o].

3. fejezet

Megoldások keresése a Matlab-bal

A 1.3 alfejezetben megmutattam, hogy a vizsgált polinom-differenciálegyenlet tetszőleges n-re visszavezethető egy egyenletrendszer megoldására. Ebben a fejezetben célom, hogy az általam a *Matlab* programcsomagban implementált *Equation_sys_solver* és *Equation_sys_solver_tester* programok segítségével megoldásokat keressek az előző fejezetben bemutatott Newton-, Broyden-, és a folytatás módszerével. Illetve a *tesztgrafikon* program segítségével szemléltetem, hogy az egyes módszerek átlagosan hány lépésből konvergálnak a kezdeti közelítéstől függően.

3.0.3. Példa. (n=2 eset)

$$\begin{aligned} p(z) &= z^2 + b \\ q(z) &= z^3 + dz + e \\ 3p'q - 2pq' &= 1 \end{aligned}$$

p és q behelyettesítése után a következőt kapjuk

$$(4d - 6b) \cdot z^2 + 6e \cdot z - 2bd = 1$$

Ebből $e = 0$ rögtön adódik. A többi változóra pedig a következő egyenletrendszer írható fel

$$\begin{aligned} 2d - 3b &= 0 \\ bd + 0.5 &= 0 \end{aligned} \tag{3.1}$$

Az *Equation_sys_solver_input_generator*-ban az egyenletrendszert function handleként tudjuk megadni az alábbi módon.(Részletesebben lásd: Függelék)

```
%szakdolgozat polinom-differenciálegyenlet: n=2,K=1
z0=[complex(3,4);complex(2,2)];
valtozok=['b';'d'];
f=@(z)([2*z(2)-3*z(1); z(1)*z(2)+0.5]);
%J=@(z)([-3 2; z(2) z(1)]);%Jacobi-mátrix
```

3.1. ábra. Equation_sys_solver_input_generator: n=2,K=1

Ekkor az *Equation_sys_solver*-t hívjuk meg a Newton-módszerre és $z0 = (3+4i, 2+2i)$ kezdeti közelítésre.

```
>> Equation_sys_solver('Newton');
Az iteráció konvergált.
Megtett iterációk száma:10
Az elért pontosság:1.1904e-012
Közelítő gyök(ök):
b:
  -0.0000 + 0.5774i

d:
  -0.0000 + 0.8660i
```

3.2. ábra. Newton: n=2,K=1

Tehát a kapott megoldás a következő:

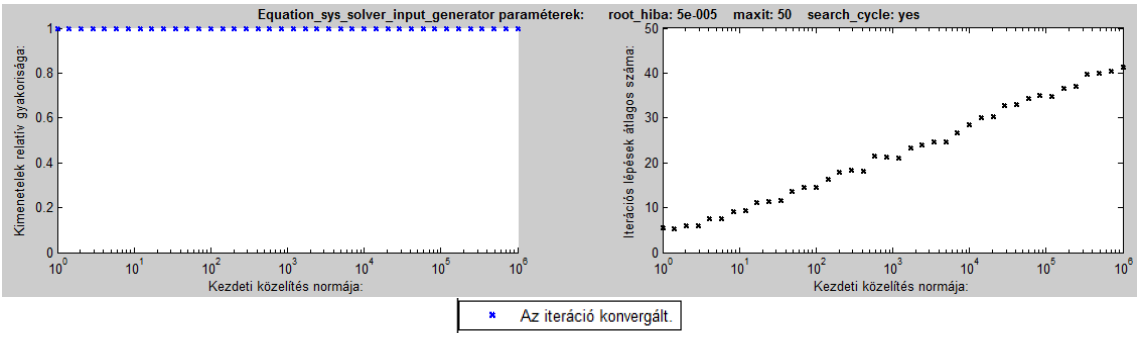
$$\begin{aligned} p(z) &= z^2 + 0.5774i \\ q(z) &= z^3 + 0.8660i \cdot z \end{aligned} \tag{3.2}$$

Vegyük észre, hogy ezek megegyeznek a 1.1.2 példában kiszámolt triviális megoldással.

Most a *tesztgrafikon* segédprogrammal fogom vizsgálni, hogy adott normakorlátok közötti véletlen kezdeti közelítésekre milyen gyakorisággal tudnak konvergálni az egyes módszerek, illetve átlagosan hány iterációs lépéssel érik el a kívánt pontosságot.

Először az 3.1-es egyenletrendszer megoldásait keresem Newton-módszerrel 10^{-5}-es pontosságra és a maximális iterációszámot futásonként 50-re korlátozva.

```
>> tesztgrafikon('Newton','complex',0,1000000,40,20,'log');
Minden algoritmus sikeresen lefutott. Futásidő:5.2443 másodperc.
A összes megtett iterációs lépés száma: 17698
```

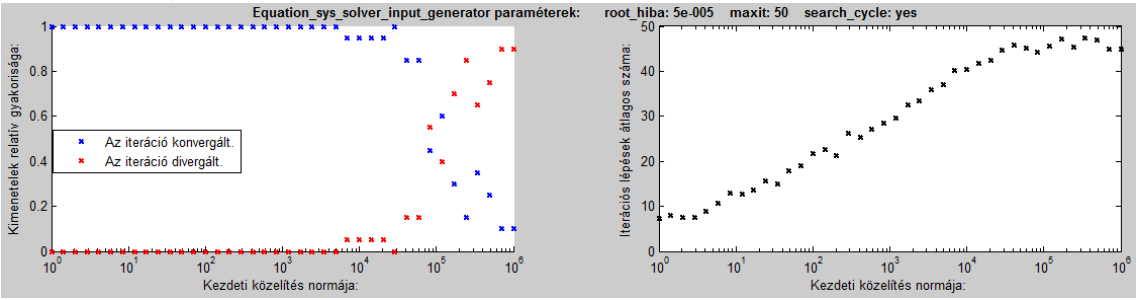


3.3. ábra. Newton: n=2, K=1

Az ábráról leolvasható, hogy $n = 2$ esetben nagy valószínűséggel konvergálni fog a módszerünk, ha tetszőleges kezdeti közelítésből indítjuk. Ez nem zárja ki az olyan tartományok létezését, ahonnan nem konvergál a módszer. Továbbá a kezdeti közelítés abszolút értékének növelésével az iterációs lépések száma is nő.

Ugyanezt vizsgáljuk meg a Broyden-módszerre is:

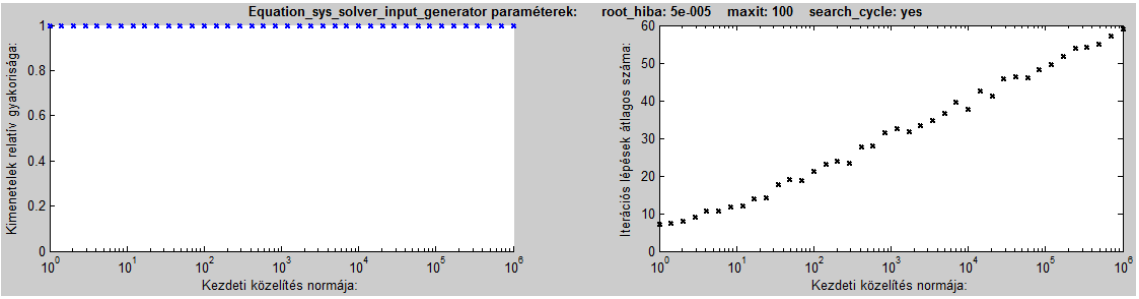
```
>> tesztgrafikon('Broyden','complex',0,1000000,40,20,'log');
Minden algoritmus sikeresen lefutott. Futásidő:7.0388 másodperc.
A összes megtett iterációs lépés száma: 23858
```



3.4. ábra. Broyden: n=2, K=1, maxit=50

A grafikonról leolvasható, hogy a Broyden-módszer átlagos iterációszáma nagyobb, mint a Newton-módszeré, és $\|z_0\| > 10^4$ esetben a módszer több esetben is divergál. Érdemes megvizsgálni, hogy megszűntethető-e a divergencia a megengedett maximális iterációszám növelésével. A maximális iterációszámot 100-ra növelve minden esetben konvergál a módszer:

```
>> tesztgrafikon('Broyden','complex',0,1000000,40,20,'log');
Minden algoritmus sikeresen lefutott. Futásidő:7.5553 másodperc.
A összes megtett iterációs lépés száma: 24705
```



3.5. ábra. Broyden: n=2, K=1, maxit=100

Most keressük meg a példánk polinom-differenciálegyenletének minél több megoldását a [0,100000] intervallumba eső normájú véletlen kezdeti közelítésekre az *Equation_sys_solver_tester*-rel:

```
>> Equation_sys_solver_tester('Newton','complex',0,100000,1000,'all');
Az algoritmus tesztelése sikeresen lefutott 10.0726 másodperc alatt.
Összesen 31242 iterációs lépés történt.
Összesítés (db):
Az iteráció konvergált: 1000
Tesztelési napló neve:2013_5_5_11_41_52.txt
```

3.6. ábra. Newton: n=2,K=1

A tesztelési napló tartalmának részlete:

```
összesítés (db):

Az iteráció konvergált: 1000
Átlagosan 31.242 lépésben.
A megtalált különböző gyökök száma: 2 Melyek a következők:
0.000003+-0.577350i;0.000005+-0.866025i;
-0.000000+0.577339i;-0.000000+0.866009i;
```

3.7. ábra. Newton: n=2,K=1

A tesztelés alapján az egyik megoldásunk 3.2, illetve megoldás lesz ennek -1-szerese is, ami természetesen adódna a 3.1 alakjából.

Érdekesség, hogy ebben az esetben a Csebisev-polinomok transzformálásából kapott megoldások egybeesnek a triviális megoldással. A *csebisev_mo_tester(2,1) Maple* függvény hívásával ez könnyen látható.

"Csebisev-polinomok:"

$$2z^2 - 1$$

$$4z^3 - 3z$$

"Transzformált Csebisev-polinomok:"

$$p = z^2 - 0. + 0.5773502690\ I$$

$$q = z^3 + (0. + 0.8660254040\ I)\,z$$

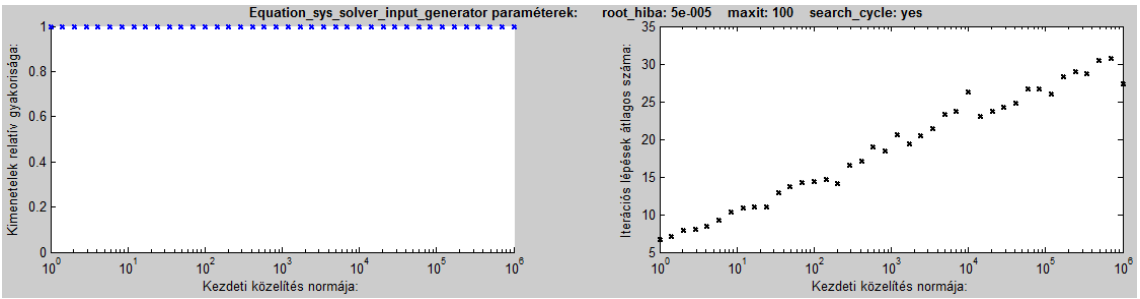
$$p = z^2 - 0. - 0.5773502690\ I$$

$$q = z^3 + (0. - 0.8660254040\ I)\,z$$

3.8. ábra. csebisev_mo_tester output: n=2,K=1

3.0.4. Példa. (n=3 eset) A 1.3.1 példában $n = 3$ és $K = 1$ esetben már kiszámoltam, hogy az 1.14-es egyenletrendszerre lehet visszavezetni a polinom-differenciálegyenletet. Erre az 5 változós egyenletrendszerre hívom meg a *tesztgrafikon*-t a Newton- és Broyden-módszerre egyaránt:

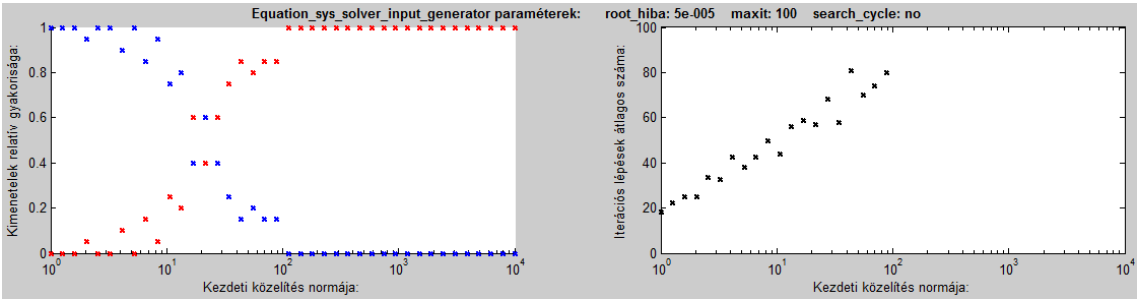
```
>> tesztgrafikon('Newton','complex',0,1000000,40,20,'log');
Minden algoritmus sikeresen lefutott. Futásidő:18.1647 másodperc.
A összes megtett iterációs lépés száma: 15018
```



3.9. ábra. Newton: n=3, K=1

A Broyden-módszert csak a $[0, 10000]$ intervallumra hívom meg, továbbá kikapcsolom a cikluskeresést, amivel futásidőt tudunk megspórolni.(Részletes beállítások lásd: Függelék)

```
>> tesztgrafikon('Broyden','complex',0,10000,40,20,'log');
Minden algoritmus sikeresen lefutott. Futásidő:23.5449 másodperc.
A összes megtett iterációs lépés száma: 63835
```



3.10. ábra. Broyden: n=3, K=1

A grafikonokról leolvasható, hogy a Newton-módszer ebben az esetben is nagy valószínűséggel konvergál, méghozzá úgy hogy az átlagos iterációk száma nem nőtt. Ezzel szemben a Broyden-módszer már $\|z_0\| > 20$ esetén jelentős relatív gyakorisággal divergál. A megengedett maximális iterációszám további növelésével itt is biztosítható a konvergencia, de a tesztelés részletezésétől most eltekintek.

Ismét használjuk a *Equation_sys_solver_tester*-t a megoldások keresésére.

```
>> Equation_sys_solver_tester('Newton','complex',0,100000,1000,'all');
Az algoritmus tesztelése sikeresen lefutott 45.1117 másodperc alatt.
Összesen 26134 iterációs lépés történt.
Összesítés (db):
Az iteráció konvergált: 1000
Tesztelési napló neve:2013_5_5_12_1_11.txt
```

3.11. ábra. Newton: n=3,K=1

A tesztelési napló tartalmának részlete:

```
Összesítés (db):

Az iteráció konvergált: 1000
Átlagosan 26.134 lépésben.
A megtalált különböző gyökök száma: 5 Melyek a következők:
0.000000+0.000000i;0.000000+0.500000i;0.000000+0.000000i;0.000000+0.666667i;0.000000+0.000000i;
1.040042+0.000000i;-0.000000+0.000000i;1.386723+0.000000i;-0.000000+0.000000i;0.240375+0.000000i;
-0.520021+0.900703i;0.000000+0.000000i;-0.693361+1.200937i;0.000000+0.000000i;-0.120187+-0.208171i;
-0.520022+-0.900702i;0.000000+0.000000i;-0.693362+-1.200936i;0.000000+0.000000i;-0.120187+0.208171i;
-0.000000+0.000000i;0.000000+-0.500000i;-0.000000+0.000000i;0.000000+-0.666667i;-0.000000+-0.000000i;
```

3.12. ábra. Newton: n=3,K=1

A triviális megoldásunk a 1.1.3 megjegyzésnek megfelelve itt is egy komplex megoldás:

$$p_{1_1}(z) = z^3 + 0.5i$$
$$q_{1_1}(z) = z^4 + \frac{2}{3}i \cdot z$$

Természetesen az egyenletrendszer alakjából következik, hogy az 1.3.1 példa jelöléseit használva a c, f változók negálhatók, így az alábbi is egy triviális megoldás:

$$p_{1_2}(z) = z^3 - 0.5i$$
$$q_{1_2}(z) = z^4 - \frac{2}{3}i \cdot z$$

Illetve a Csebisev-polinomokból transzformált megoldásaink megegyeznek a 1.1.9 példában a *Maple*-ben kiszámolt polinomokkal, melyek közül az egyik valós együtthatós megoldás (1.7). Érdemes megvizsgálni, hogy a *Equation_sys_solver_tester* valós kezdeti közelítésekből indulva valóban csak ezt az egy megoldást fogja megtalálni.

```
>> Equation_sys_solver_tester('Newton','real',0,10000,100,'all');
Az algoritmus tesztelése sikeresen lefutott 2.5899 másodperc alatt.
Összesen 2510 iterációs lépés történt.
Összesítés (db):
Az iteráció konvergált: 97
Az iteráció divergált: 3
Tesztelési napló neve:2013_5_3_23_21_51.txt
```

3.13. ábra. Newton: n=3,K=1

A tesztelési napló tartalmának részlete:

```
Összesítés (db):

Az iteráció konvergált: 97
Átlagosan 24.3299 lépésben.
A megtalált különböző gyökök száma: 1 Melyek a következők:
1.040042+0.000000i;-0.000000+0.000000i;1.386723+0.000000i;-0.000000+0.000000i;0.240375+0.000000i;


Az iteráció divergált: 3
1886.694957+0.000000i;-4737.320298+0.000000i;-85.068290+0.000000i;9828.419703+0.000000i;-7308.692224+0.000000i;
-751.180479+0.000000i;3280.951407+0.000000i;2919.964105+0.000000i;-3122.239068+0.000000i;1787.663594+0.000000i;
-620.775898+0.000000i;-4693.713231+0.000000i;3490.774862+0.000000i;-7509.109423+0.000000i;-4964.686031+0.000000i;
```

3.14. ábra. Newton: n=3,K=1

A Newton–Raphson-módszer valóban csak a valós megoldáshoz konvergált, viszont 3-szor divergált. Ennek oka lehet, hogy egyes esetekben több iterációs lépés kell a konvergenciához csak valós közelítésekre futtatva, mint a komplex közelítések esetén. Tehát növeljük a megengedett maximális iterációk számát 100-ra és nézzük meg, hogy így vajon konvergál-e a Newton–Raphson-módszer az alábbi kezdeti közelítésre.

$$z0 = [1886.694957; -4737.32029; -85.068290; 9828.419703; -7308.692224]$$

```
>> Equation_sys_solver('Newton');
Az iteráció konvergált.
Megtett iterációk száma:52
Az elért pontosság:2.9057e-005
Közelítő gyök(ök):
a:
    1.0400

b:
 -3.8491e-006

c:
    1.3867

d:
 -5.1322e-006

e:
    0.2404

Futásidő:0.1255 másodperc
```

3.15. ábra. Newton: n=3,K=1

Azaz a módszer a várt valós megoldáshoz konvergált.

3.0.5. Példa. (n=4) Továbbra is $K := 1$ választással a polinom differenciál-differenciálegyenletünk a következő:

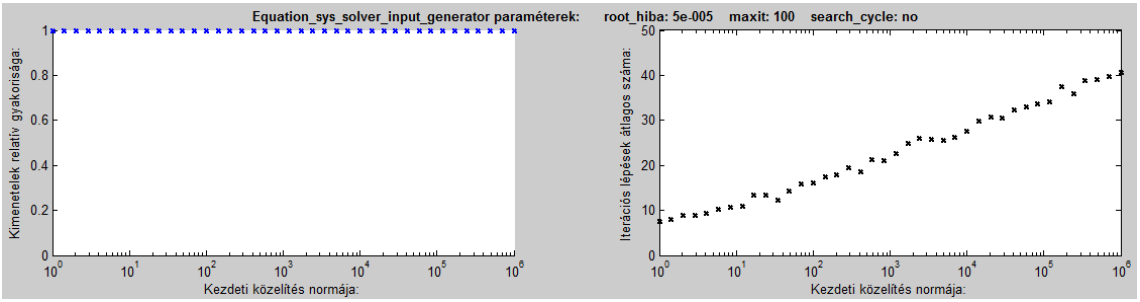
$$p(z) := z^4 + az^2 + bz + c$$
$$q(z) := z^5 + dz^3 + ez^2 + fz + g$$
$$5 \cdot p'q - 4 \cdot pq' = 1$$

Ez az alábbi egyenletrendszerre vezethető vissza:

$$\begin{aligned} -10a + 8d &= 0 \\ -15b + 12e &= 0 \\ -2ad + 16f - 20c &= 0 \\ -7bd + 2ae + 20g &= 0 \\ -3be + 6af - 12cd &= 0 \\ bf + 10ag - 8ce &= 0 \\ 5bg - 4cf - 1 &= 0 \end{aligned}$$

A konvergencia-tulajdonságot, és az átlagos iterációk számát most csak a Newton-módszerre vizsgálom:

```
>> tesztgrafikon('Newton','complex',0,1000000,40,20,'log');
Minden algoritmus sikeresen lefutott. Futásidő:60.7079 másodperc.
A összes megtett iterációs lépés száma: 18190
```



3.16. ábra. Newton: n=4, K=1

A grafikonról azt olvashatjuk le, hogy az átlagos iterációk száma megnőtt az előző példához képest.

A megoldásokat ismét az *Equation_sys_solver_tester*-rel keressünk:

```
>> Equation_sys_solver_tester('Newton','complex',0,100000,1000,'all');
Az algoritmus tesztelése sikeresen lefutott 152.2673 másodperc alatt.
Összesen 31880 iterációs lépés történt.
Összesítés (db):
Az iteráció konvergált: 1000
Tesztelési napló neve:2013_5_5_12_21_29.txt
```

3.17. ábra. Newton: n=4,K=1

A tesztelési napló tartalmának részlete:

```
Összesítés (db):

Az iteráció konvergált: 1000
Átlagosan 31.88 lépésben.
A megtalált különböző gyökök száma: 14 Melyek a következők:
0.617564+-0.617564i;0.533207+0.220861i;-0.000000+-0.286039i;0.771955+-0.771955i;0.666508+0.276077i;-0.000000+-0.476731i;0.145527+-0.060279i;
0.617565+0.617563i;-0.533206+0.220861i;0.000001+0.286039i;0.771956+0.771954i;-0.666508+0.276076i;0.000001+0.476731i;-0.145527+-0.060279i;
1.124683+1.124683i;-0.000000+0.000000i;0.000000+0.316228i;1.405853+1.405853i;-0.000000+0.000000i;0.000000+0.790569i;-0.000000+-0.000000i;
-1.124683+-1.124684i;-0.000000+-0.000000i;-0.000000+0.316228i;-1.405854+-1.405855i;-0.000000+-0.000001i;-0.000000+0.790570i;0.000000+0.000000i;
0.000001+-0.000002i;-0.000001+-0.000000i;0.000000+-0.447213i;0.000001+-0.000002i;-0.000001+-0.000000i;0.000000+-0.559017i;-0.000000+0.000000i;
-1.124683+1.124683i;-0.000000+-0.000000i;-0.000000+-0.316228i;-1.405853+1.405854i;-0.000000+-0.000000i;-0.000000+-0.790569i;0.000000+-0.000000i;
0.000000+0.000000i;0.000000+-0.000000i;-0.000000+0.447214i;0.000000+0.000000i;0.000000+-0.000000i;0.000000+0.559017i;0.000000+-0.000000i;
0.617564+-0.617564i;-0.533207+-0.220861i;-0.000000+-0.286039i;0.771955+-0.771955i;-0.666508+-0.276077i;-0.000000+-0.476731i;-0.145527+0.060279i;
0.617566+0.617568i;0.533202+-0.220861i;-0.000000+0.286039i;0.771958+0.771960i;0.666503+-0.276076i;-0.000000+0.476733i;0.145526+0.060278i;
1.124683+-1.124683i;0.000000+-0.000000i;0.000000+-0.316228i;1.405853+-1.405853i;0.000000+-0.000000i;0.000000+-0.790569i;-0.000000+-0.000000i;
-0.617564+-0.617564i;-0.220861+-0.533207i;0.000000+0.286039i;-0.771955+-0.771955i;-0.276077+-0.666508i;0.000000+0.476731i;-0.060279+0.145527i;
-0.617564+0.617564i;0.220861+-0.533207i;-0.000000+-0.286039i;-0.771955+0.771955i;0.276077+-0.666508i;-0.000000+-0.476731i;0.060279+0.145527i;
-0.617564+-0.617564i;0.220861+0.533207i;0.000000+0.286039i;-0.771954+-0.771955i;0.276077+0.666509i;0.000000+0.476731i;0.060279+-0.145527i;
-0.617564+0.617564i;-0.220861+0.533207i;-0.000000+-0.286039i;-0.771955+0.771955i;-0.276077+0.666508i;-0.000000+-0.476731i;-0.060279+-0.145527i;
```

3.18. ábra. Newton: n=4,K=1

Ezek három lényegesen eltérő megoldást takarnak:
A triviális megoldás:

$$p_1(z) = z^4 + 0.447214i$$
$$q_1(z) = z^5 + 559017i \cdot z$$

A Csebisev-polinomból transzformált egyik megoldás:

$$p_2(z) = z^4 + (1.124683 - 1.124683i) \cdot z^2 - 0.316228i$$
$$q_2(z) = z^5 + (1.405853 - 1.405853i) \cdot z^3 - 0.790569i \cdot z$$

A *Equation_sys_solver_tester* naplójából kiolvashatjuk, hogy a Newton–Raphson-módszer mind a négy Csebisev-polinomokból transzformált megoldást is találta (3., 4., 6., 10. megoldások a naplóban). Ezt a korábbi példákhoz hasonlóan ellenőrizhetjük a *Maple*-ben implementált *csebisev_mo_tester* függvénnyel.

A harmadik megoldásunk, ami a polinomok alakja miatt nem sorolható se a triviális, se a Csebisev-polinomokból transzformált megoldások közé az alábbi.

$$p_2(z) = z^4 - (0.61 + 0.61i) \cdot z^2 - (0.22 + 0.53i) \cdot z + 0.28i$$
$$q_2(z) = z^5 - (0.77 + 0.77i) \cdot z^3 - (0.27 + 0.66i) \cdot z^2 + 0.47i \cdot z + (-0.06 + 0.14i)$$

3.0.6. Példa. (Folytatás módszere) A 2.1.6-es tétel kimondta, hogy egy megfelelő $f : \mathbb{R}^n \to \mathbb{R}^n$ leképezésre, ha a Newton-módszer nem konvergálna egy valós kezdeti közelítésre, de létezik valós megoldás, és teljesülnek a 2.1.6-es tétel feltételei, akkor érdemes használni a folytatás-módszerét.

Mivel a fenti példákban a Newton-módszer mindig konvergált, ezért ebben a példában a maximális iterációszám csökkentésével próbálom megtalálni az 3.0.4-es példában vizsgált polinom-differenciálegyenlet ($n = 3, K = 1$) egy nem feltétlenül valós megoldását, ezzel szemléltetve a módszer azon jó tulajdonságát, hogy lokális lépések sorozatával próbálja megtalálni az eredeti megoldást. A fent említett tétel nem garantálja, hogy a módszer komplex kezdeti közelítésre is használható lesz, azonban a példában használt paraméterekre ez épp teljesülni fog.

Legyen a kezdeti közelítésünk az alábbi, illetve követeljünk meg
$root\_hiba = 5.000e - 008$ pontosságot.

$$z0 = [1 + 2i; 0.5 + 0.5i; -1 - 0.8i; -i; 2 + 0.5i] \cdot 100$$

A 3.9-es ábráról leolvashatjuk, hogy ekkor a Newton-módszer átlagosan 15 lépésben konvergált, ezért korlátozzuk a maximális iterációszámot 12-re. Ekkor azt tapasztaljuk, hogy erre a kezdeti közelítésre már nem konvergál a Newton-módszer.

```
>> Equation_sys_solver('Newton');
Az iteráció divergált!
Futásidő:0.057103 másodperc
```

3.19. ábra. Newton: n=3,K=1

Hívjuk meg a folytatás módszerét:

```
A folytatás módszere megtalálta az eredeti megoldást k=4-ra.
Az iteráció konvergált.
Megtett iterációk száma:75
Az elért pontosság:3.1083e-013
Közelítő gyök(ök):
a:
  -0.5200 - 0.9007i

b:
  1.5894e-014 +3.5285e-014i

c:
  -0.6934 - 1.2009i

d:
  2.1191e-014 +4.7046e-014i

e:
  -0.1202 + 0.2082i

Futásidő:0.3804 másodperc
```

3.20. ábra. Folytatás módszere: n=3,K=1

Azaz a Folytatás-módszere $k = 4$-szer futtatott köztes Newton-módszert, melyek mindegyike lefeljebb 12 iterációs lépés után konvergált a köztes lépésekhez tartozó módosított egyenletrendszerre, és végül pont az eredeti egyenletrendszer egyik Csebisev-transzformált megoldását találta meg. A folytatás módszere az implementálásból adódóan a megfelelő $k = 4$ érték megtalálásáig több iterációs lépést is végrehajtott, és csak ezután tudott lefutni a konvergencia, így kaphattunk 75-öt a megtett iterációk számára. Ezzel összességében azonban sokkal több iteráció történt, mint ha engedtük volna a Newton-módszert elsőre konvergálni.

```
>> Equation_sys_solver('Newton');
Az iteráció konvergált.
Megtett iterációk száma:15
Az elért pontosság:8.7116e-010
Közelítő gyök(ök):
a:
  -0.5200 - 0.9007i

b:
  8.2427e-012 -1.0802e-010i

c:
  -0.6934 - 1.2009i

d:
  1.0990e-011 -1.4403e-010i

e:
  -0.1202 + 0.2082i

Futásidő:0.040742 másodperc
```

3.21. ábra. Newton: n=3,K=1

3.0.7. Megjegyzés. A példabeli $maxit = 12$ választás épp megfelelőnek bizonyult a megadott kezdeti közelítéshez abból a szempontból, hogy a folytatás módszere egymás utáni

lokális konvergenciákkal végül megtalálta az eredeti feladat megoldását, méghozzá nem túl sok köztes Newton-módszert felhasználva. Ez nem feltétlenül teljesül mindig.

pl.:

$$z0 = [1 + 2i; 0.5 + 0.5i; -1 - 0.8i; -i; 2 + 0.5i] \cdot 1000$$

kezdeti közelítésre, és továbbra is $maxit = 12$ választással nincs alkalmas $k < 100$ paraméter.

```
>> Equation_sys_solver('FolytN');
A folytatás módszere nem talált megfelelő k kiindulási paramétert az induláshoz!
Futásidő:96.5071 másodperc
```

3.22. ábra. Folytatás módszere: n=3,K=1

Összefoglalás

A dolgozatban célom az 1.1 megoldásainak numerikus meghatározása volt $n = \{2, 3, 4\}$, $K = 1$ esetben. Az első fejezetben két általános megoldás alakját határoztam meg. A triviális eset a polinom-differenciálegyenletbe való helyettesítésből könnyen adódott. Együtthatóinak képzetessége a K nemnulla konstans előjelétől függ (1.1.3 megjegyzés). Ezután több állításon keresztül (1.1.6, 1.1.7, 1.1.8) sikerült megadni azt a transzformációt, amivel tetszőleges n-re az n- és $(n+1)$-edfokú Csebisev-polinomokból előállíthatók az 1.1 p,q megoldásai. A megoldások közönséges differenciálegyenlettel történő vizsgálata után a 1.1.9-es példán keresztül megmutattam, hogy a vizsgált polinom-differenciálegyenlet megoldásai hogyan használhatók fel primitív függvény keresésére az adott esetben.

A vizsgált polinom-differenciálegyenlet egy nemlineáris egyenletrendszerre vezethető vissza, ahol minden egyenlet a p,q együtthatóinak polinomiális függvénye (1.3 fejezet). A egyenletrendszer megoldására a Newton–Raphson-módszert használtam. Adott feltételeket kielégítő vektormezők esetén a módszer lokálisan, és monoton is tud konvergálni (2.1.2, 2.1.3 tételek), sőt a folytatás módszere a módszer globális konvergenciát is tudja biztosítani (2.1.6). Azonban általános $f : \mathbb{C}^n \to \mathbb{C}^n$ leképezések esetén csak $n = 1$ esetén vannak ismert eredmények a Newton–Raphson-módszer globális konvergencia tulajdonságairól. Ezek azt mutatják, hogy az esetünkben is érdemes ezzel a módszerrel vizsgálni a polinom-differenciálegyenletből levezethető egyenletrendszert. A második fejezetben egy kvázi-Newton-módszer, a Broyden-módszer is bemutatásra került, ami a Shermann–Morrison formula felhasználásával minden iterációs lépésben rögtön a Jacobi-mátrix közelítésének inverzét tudja meghatározni.

A harmadik fejezetben több példán keresztül írom le az $n = \{2, 3, 4\}, K = 1$ esetek numerikus megoldásait, illetve megmutatom, hogy a Broyden-módszer a változók számának növelésével a Newton-módszerhez képest sokkal több iterációs lépésben tud csak konvergálni. A példákban külön kiemeltem azon megoldásokat, melyek az első fejezetben részletezett triviális, illetve Csebisev-polinomokból transzformált megoldásokat jelentik. A tesztelések során azt tapasztaltam, hogy a Newton–Raphson-módszer rendkívül jól használható a polinom-differenciálegyenletből levezethető egyenletrendszer megoldására, hiszen a módszer a vizsgált n értékekre viszonylag kevés iterációs lépés után minden véletlen kezdeti közelítésre konvergált.

Végül a 3.0.6 példában egy speciális inputra, a Newton-módszer maximális iterációszámát jelentősen lekorlátozva a folytatás módszere több lokális konvergenciával eljutott

a keresett megoldáshoz, összesen azonban jóval több iterációs lépéssel, mintha a Newton-módszert futtattam volna kellően nagy megengedett maximális iterációszámra. Tehát a szakdolgozatban vizsgált három módszer közül a Newton–Raphson-módszer bizonyult a leghatékonyabbnak a vizsgált polinom-differenciálegyenlet megoldásainak numerikus meghatározására az $n = \{2, 3, 4\}$, $K = 1$ esetekre.

Irodalomjegyzék

[1] ISTVÁN SIGRAY, *Solution of the polynomial equation* $kp'(z)q(z) - lp(z)q'(z) = cp^m(z)$, Studia Scientiarum Mathematicarum Hungarica 45 (2), 161-195

[2] STOYAN GISBERT, TAKÓ GALINA, *Numerikus módszerek 1.,Typotex kiadó, Budapest, 2002*

[3] KERÉNYI PÉTER, *Gyökkeresés iterációval, BSc szakdolgozat, Budapest, 2011* http://www.cs.elte.hu/blobs/diplomamunkak/bsc_alkmat/2011/kerenyi_peter.pdf

[4] WIKIPÉDIA, Chebyshev polynomials, http://en.wikipedia.org/wiki/Chebyshev_polynomials

[5] WIKIPÉDIA, Newton's method, http://en.wikipedia.org/wiki/Newton%27s_method

[6] WIKIPÉDIA, Broyden's method, http://en.wikipedia.org/wiki/Broyden%27s_method

[7] WIKIPÉDIA, Jacobian conjecture http://en.wikipedia.org/wiki/Jacobian_conjecture

A. Függelék

Leírás a szakdolgozathoz készített programokhoz

A.1. csebisev_mo_tester

Ez a *Maple*-ben megírt függvény a 1.1.8-es állításban meghatározott képletek alapján számolja ki az 1.1-es polinom-differeciálegyenlet Csebisev-polinomok transzformálásából kapott megoldásait.

Programkód:

```
csebisev_mo_tester:=proc(n,K)
p:=collect(csebisev(n,z,z),z);
q:=collect(csebisev(n+1,z,z),z);
dp:=diff(p,z);
dq:=diff(q,z);
diffeq1:=collect((n+1)*dp*q-n*dq*p,z);
#print(diffeq1);
print("Csebisev-polinomok:");
print(p);
print(q);
b:=solve(w^(2*n)=-n*(n+1)/(K*2^(2*n-1)),w);
print(\n);
print("Transzformált Csebisev-polinomok:");
for i from 1 to n do
x:=b[i]*z;
pt:=evalf(collect(csebisev(n,x,z)/(2^(n-1)*b[i]^n),z));
qt:=evalf(collect(csebisev(n+1,x,z)/(2^(n)*b[i]^(n+1)),z));
dpt:=diff(pt,z);
dqt:=diff(qt,z);
diffeq2:=collect((n+1)*dpt*qt-n*dqt*pt,z);
#print(diffeq2);
print('p'=evalf(pt));
print('q'=evalf(qt));
print(\n);
od;
end proc:
```

```
csebisev:=proc(n,x,z)
p0:=1;
p1:=x;
if n=0 then
return(p0);
elif n=1 then
return(p1);
else
for i from 2 to n do
p:=2*x*p1-p0;
p0:=p1;
p1:=p;
od;
return(collect(p,z));
fi;
```

A.2. Equation_sys_solver

Ezt a programot akkor érdemes használni, ha arra vagyunk kíváncsiak, hogy adott kezdeti közelítésre milyen eredménnyel fut le a fent említett módszerek valamelyike.

Paraméterek megadása

Parancssor: *Equation_sys_solver*(*algorithm*)

- *algorithm*='Newton','Broyden','FolytN' : itt tudjuk megadni, hogy melyik algoritmussal akarjuk megoldani az egyenletrendszert.

Minden további paramétert az *Equation_sys_solver_input_generator* m-fájl felületén tudunk megadni:

Futást befolyásoló paraméterek:

- *root_hiba* : mekkora pontosságot akarunk elérni
- *maxit* : megengedett maximális iterációk száma
- *display*='all'/'res+comment'/'res+z0'/'res'/'no' : különböző megjelenítési funkciók. (Mindent kiír a képernyőre/ csak eredményt+megjegyzést/ csak eredményt és a kezdeti közelítést/ csak az eredményt/ semmit).
- *search_cycle*='yes'/'no' : a cikluskeresés bekapcsolása opcionális.

Adott egyenletrendszerhez tartozó input:

- *z0* : kezdeti közelítés
- f : az egyenletrendszer megadása function handle-ben
- J : Newton-módszer esetén a Jacobi-mátrixot megadhatjuk function handle-ként. Nem szükséges ezt megtennnünk, hiszen a program ezt automatikusan közelíti a 2.2.1 fejezetben leírtak alapján, ha nincs megadva ilyen J function handle.
- *valtozok* : az ismeretlenek vektorát jelölik. Az implementáció miatt fontos, hogy minden változó azonos számú karakterből álljon!

```
%Paraméterek megadása:
root_hiba=5.000e-005;%Kívánt pontosság
maxit=50; %megengedett maximális iterációszám
display='res';%output funkciók?('all'/'res+comment'/'res+z0'/'res'/'no')
search_cycle='yes';%cikluskeresés bekapcsolása ('yes'/'no')

%FONTOS: a változok vektor minden eleme uannyi karakterből álljon (pl.:a001,...,a100)
%Induló érték, és egyenletrendszer megadása:

%szakdolgozat polinom-differenciálegyenlet: n=3,K=1
z0=[complex(1,2);complex(0.5,0.5);complex(-1,-0.8);complex(0,-1);complex(2,0.5)].*1000;
valtozok=['a';'b';'c';'d';'e'];
f=@(z)([3*z(3)-4*z(1);3*z(4)-4*z(2);6*z(5)-z(1)*z(3);z(1)*z(4)-6*z(2)*z(3);-3*z(2)*z(4)+4*z(1)*z(5)-1]);
```

A.1. ábra. Minta Equation_sys_solver_input_generator tartalom

Az algoritmusok lehetséges futásai

A lehetséges outputokat különböző *exit_code*-ok jelölik:

- *exit_code=0* : Az iteráció konvergált.
- *exit_code=1* : Az iteráció divergált.
- *exit_code=2* : Az iteráció ciklusba futott.
- *exit_code=3* : A Jacobi-mátrix (közelítése) szinguláris.
- *exit_code=4* : A folytatás módszere nem talált a rögzített korlátnál kisebb megfelelő k kiindulási paramétert az induláshoz.

Megoldó algoritmusok

Mindhárom algoritmus esetén az input paraméterek a fentieknek felelnek meg.

Output paraméterek:

- *exit_code* : az algoritmus futását jellemzi.
- *roots* : ha az iteráció konvergált, akkor a megoldásdást tárolja
- *ciklustomb* : ha az iteráció ciklizál, akkor a ciklust tárolja
- *it* : megtett iterációk száma
- *diff* : az utolsó iterációs közelítés normáját tartalmazza

```
[exit_code,roots,it,ciklustomb,diff]=Newton_it(z0,root_hiba,f,J,maxit,search_cycle)
it=0;
bool_ciklus=0;hamis
ciklustomb=[ ];
roots=[ ];
if search_cycle then
    Az iterációs közelítéseket tartalmazó mátrixban z0 eltárolása
end
diff=norm(f(z0),2);
while it<maxit && diff>root_hiba && bool_ciklus==0 do
    if Meg volt adva a J function handle then
        M=J(z0);
    else
        M=Jacobi_approx(z0,f);
    end
    if M szinguláris then
        kilépés 3-as hibakóddal
    else
        zz=M\(-f(z0));
        zn=z0+zz;
        diff=norm(f(zn),2);
        it=it+1;
        z0=zn;
        if search_cycle then
            Az iterációs közelítéseket tartalmazó mátrixban zn eltárolása
            Sor_ism_ell függvénnyel cikluskeresés, és bool_ciklus=1, ha találtunk
            ciklust.
        end
    end
end
Kilépési kódok generálása, és kilépés;
```

Algorithm 1: Newton-módszer implementálása

```
[exit_code,roots,it,ciklustomb,diff]=Broyden_it(z0,root_hiba,f,maxit,search_cycle))
it=0;
bool_ciklus=0;
ciklustomb=[ ];
roots=[ ];
if search_cycle then
  | Az iterációs közelítéseket tartalmazó mátrixban z0 eltárolása
end
A0=Jacobi_approx(z0,f);
s=-A0\f(z0);
z1=z0+s;
it=it+1;
if search_cycle then
  | Az iterációs közelítéseket tartalmazó mátrixban z1 eltárolása
end
diff=norm(f(z1));
y=f(z1)-f(z0);
if A0 szinguláris then
  | kilépés 3-as hibakóddal
else
  | invA0=inv(A0);
end
z0=z1;
while it<maxit && diff>root_hiba && bool_ciklus==0 do
  | [invA1]=SherMorBroy(invA0,s,y); Shermann-Morrison formula
  | if A1 szinguláris then
  |   | kilépés 3-as hibakóddal
  | else
  |   | s=-invA1*f(z0);
  |   | zn=z0+s;
  |   | it=it+1;
  |   | diff=norm(f(zn));
  |   | if search_cycle then
  |   |   | Az iterációs közelítéseket tartalmazó mátrixban zn eltárolása
  |   |   | Sor_ism_ell fügvénnyel cikluskeresés, és bool_ciklus=1 ha találtunk
  |   |   | ciklust
  |   | end
  |   | y=f(zn)-f(z0);
  |   | z0=zn;
  |   | invA0=invA1;
  | end
end
Kilépési kódok generálása, és kilépés;
```

Algorithm 2: Broyden-módszer implementálása

Folyt_Newton

$[exit\_code, roots, sum\_it, ciklustomb, diff, k] = Folyt\_Newton(z0, root\_hiba, f, J, maxit, search\_cycle)$

A k output paraméter jelöli, hogy a folytatás módszere hányszor futtatja a köztes módosított Newton-iterációt, ha létezik ez a megfelelő k. Az algoritmus implementálásának vázolásától itt most eltekintek.

A.3. Equation_sys_solver_tester

Akkor érdemes használni, ha arra vagyunk kiváncsiak hogy adott normakorlátok között több véletlen kezdeti közelítésből is elindítva egy adott módszert, az hányszor konvergál, divergál stb. Melyek lesznek a teszt során kapott megoldások, illetve azok a kezdeti kiindulások, melyekre nem konvergált. A tester a konvergens esetek különböző gyökeit is eltárolja.

Paraméterek megadása

Parancssor: $Equation\_sys\_solver\_tester(algorithm, mode, lb, ub, times, testdisplay)$

- *algorithm*='Newton','Broyden'
- *mode*='complex','real' : csak a valós, vagy a komplex értékek között futtatjuk a módszert. A 'real'-mód használatával tudjuk keresni a csak valós megoldásokat, ha egyáltalán létezik valós megoldás.
- *lb* : alsó normakorlát
- *ub* : felső normakorlát
- *times* : hányszor futtassuk a megadott módszert
- *testdisplay*='all'/'nodiary'/'no' : Lehetőség van választani a teszt eredményének részletes naplózására egy szövegfájlba *'all'*, vagy csak alapvető információk képernyőre írása között *'nodiary'*. A *'no'* funkció nem ajánlott, hiszen ekkor semmilyen látható információt nem kapunk. Ezt a funkciót a *tesztgrafikon* program használja. A naplófájl mindig a futtatás pontos dátumával kerül elmentésre.

A tester az *Equation_sys_solver_input_generator*-ban megadott f leképezésre, és $z0$-ból kinyert dimenzióra fog tesztelni. Az m-fájlok mellé a 3. fejezetben szereplő példákhoz tartozó teljes naplófájlokat is csatoltam.

A.4. tesztgrafikon

A megadott normakorlátok között a norma függvényében grafikusan szemlélteti, hogy az adott módszer többszöri véletlen kezdeti közelítésből milyen relatív gyakorisággal konvergált, divergált stb. Illetve a konvergáló esetek átlagos iterációszámát is ábrázolja a kezdeti közelítés normájának függvényében.

Paraméterek megadása

parancssor: *tesztgrafikon(algorithm,mode,lb,ub,db_nodes,minta_db,felosztas)*

- *algorithm*='Newton','Broyden'
- *mode*='complex','real'
- *lb* : alsó normakorlát
- *ub* : felső normakorlát
- *db_nodes* : hány részintervallumra legyen meghívva a tester. Kikötés:$db\_nodes \geq 2$!
- *minta_db* : egy részintervallumon belül hány véletlen kezdeti közelítésből futtasson algoritmust a tester.
- *felosztas*='ekvi','log' : a részintervallumokat reprezentáló pontok egyenletesen vagy logaritmikusan helyezkedjenek el.

Ezen kívül a *tesztgrafikon* m-fájlban lehetőségünk van a *makelegend* változóval beállítani, hogy készüljön-e jelmagyarázat a grafikonhoz.

A *tesztgrafikon* az *Equation_sys_solver_input_generator*-ban megadott f leképezésre, és $z0$-ból kinyert dimenzióra a *tesztgrafikon* paramétereitől függően fogja részintervallumonként futtatni az *Equation_sys_solver_tester*-t.

Futási idő jellemzése a függvényekben megadott paraméterek szerint

A 3. fejezetben láthattuk, hogy a Newton-módszer az általam vizsgált esetekben mindig konvergált, viszont az iterációk száma a kezdeti érték normájának növelésével nőtt. Éppen ezért ha nagy normaintervallumon akarjuk meghívni a *tesztgrafikon* függvényt, akkor *log* felosztással várhatóan hamarabb fog lefutni, mint az *ekvi* eset, hiszen az egyenletesen veszi a kezdeti közelítéseket.

A 3. fejezetben láthattuk, hogy a Broyden-módszer, ha konvergált akkor azt sokkal több iterációs lépésben tudta csak megtenni, mint a Newton-módszer. Éppen ezért ha megengednénk, hogy az algoritmus egy kezdeti közelítésre elég sokáig fusson (pl.:1000 iterációs lépés), akkor érdemes kikapcsolni a cikluskeresést, mert azzal is tudjuk csökkenteni a futásidőt.