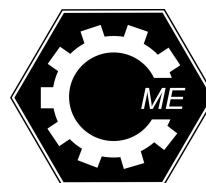




MISKOLCI EGYETEM
GÉPÉSZMÉRNÖKI ÉS
INFORMATIKAI KAR

Gép- és Terméktervezési Intézet



TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

Spektrál módszer időfüggő konvektív hőátadásra örvényvektor-áramfüggvény formalizmusban

Csáti Zoltán

II. éves MSc gépészmérnök hallgató

Konzulens:

Vadászné dr. Bognár Gabriella

egyetemi tanár

Gépészmérnöki és Informatikai Kar

Miskolc

2014

Tartalomjegyzék

1. Jelölések	3
2. Bevezetés	5
3. Alapegyenletek	6
3.1. Primitív változós formalizmus	7
3.2. Örvényvektor-áramfüggvény formalizmus	7
3.3. Egyéb formalizmusok	9
3.4. Konvektív hőátadás	10
4. Spektrál módszerek	12
4.1. Interpoláció	12
4.2. Spektrális deriválás	15
4.3. A spektrál módszerek jellemzői	16
4.4. Általánosítás négyszög tartományra	16
5. A kezdeti-peremértékfeladat numerikus megoldása	19
5.1. A Helmholtz-egyenlet megoldása Dirichlet-peremfeltételekkel	19
5.2. A Helmholtz-egyenlet megoldása Robin-peremfeltételekkel	22
5.3. Időbeli diszkretizáció	24
5.4. Térbeli diszkretizáció	27
5.5. Leállási feltétel, konvergencia	29
6. A MATLAB programok felépítése	30
6.1. Az algoritmusok megvalósításának szempontjai	35
7. Numerikus példák	37
7.1. „Lid-driven cavity”	37
7.2. Regularizált „lid-driven cavity”	41
8. Összefoglalás, további célok	44
9. Hivatkozások	45
A. MATLAB programok	47

1. Jelölések

A dolgozatban egységesen a következő jelöléseket használjuk. A skalárokat normál betűstílussal, a vektorokat félkövér kisbetűvel, a mátrixokat félkövér nagybetűvel jelöljük. A ∇ operátor – ha külön nem jelöljük – a mellette álló függvényre hat. A következő táblázat a gyakran előforduló mennyiségek megnevezését és mértékegységét tartalmazza.

Jelölés	Megnevezés	Mértékegység
<i>latin betűk</i>		
c_p	állandó nyomáson vett fajhő	$J/(molK)$
D	(első) deriváló mátrix	-
D^(k)	k -adik deriváló mátrix	-
f	tömegegységen megoszló erőrendszer intenzitásvektora	N/kg
g	nehézségi gyorsulás	m/s^2
I	egységmátrix	-
I_N	N -edfokú approximációs polinom	-
J	Jacobi-mátrix	-
k	hővezetési tényező	$W/(mK)$
L	karakterisztikus hossz	m
ℓ_j	j -edik Lagrange-féle interpolációs polinom	-
M	hatásmátrix	-
n	normális	-
p	nyomás	Pa
p_N	N -edfokú interpolációs polinom	-
$\mathbb{P}_N$	legfeljebb N -edfokú polinomok tere	-
Pr	Prandtl-szám	-
Ri	Richardson-szám	-
Re	Reynolds-szám	-
t	idő	s
T	hőmérséklet	K
T_0	referencia hőmérséklet	K
T_n	elsőfajú n -edfokú Chebyshev-polinom	-

$\mathbf{v}$	sebességvektor	m/s
u, v	a sebességvektor koordinátái	m/s
$\mathbf{x}$	pillanatnyi konfiguráció helyvektora	m
<u>görög betűk</u>		
β	térfogati hőtágulási együttható	$1/K$
$\Gamma, \partial\Omega$	tartomány pereme	m vagy m^2
κ	hőmérséklet-vezetési tényező	m^2/s
μ	dinamikai viszkozitás	$Pa\cdot s$
ν	kinematikai viszkozitás	m^2/s
ρ	sűrűség	kg/m^3
ρ_0	referencia sűrűség	kg/m^3
$\boldsymbol{\sigma}$	Cauchy-féle feszültségtenzor	N/m^2
ψ	áramfüggvény	m^2/s
ω	örvényesség	$1/s$
$\boldsymbol{\omega}$	örvényvektor	$1/s$
Ω	tartomány	m vagy m^2
<u>szimbólumok</u>		
δ_{ij}	Kronecker szimbólum	
$\otimes$	Kronecker-féle tenzorszorzat	
$\langle \cdot \rangle$	belső szorzat	
$\mathcal{O}$	nagy ordó	

1. táblázat: A dolgozatban használt jelölések

2. Bevezetés

A legtöbb áramlás a természetben és az ipari tevékenységek következtében jön létre. Ezen komplex jelenségek megértésében és a technológiai alkalmazások fejlesztése érdekében szükséges a folyadékáramlásokat modelleznünk. Mióta nagy teljesítményű számítógépek állnak rendelkezésre, a számítások egyre inkább kiegészítik és néha helyettesítik a közvetlen méréseket. Bizonyos esetekben a mérés túl drága, vagy akár nem is lehetséges, így a szimuláció az egyetlen hatékony módszer egy tervezéshez szükséges adatok meghatározásához és a szimulációkban szereplő paraméterek hatásainak elemzésére [9].

A newtoni folyadék mozgását leíró Navier-Stokes egyenletekre egzakt megoldás csak nagyon speciális esetben adható meg. A leggyakoribb megoldási módszerek közé tartoznak a véges differencia, a véges elem és a véges térfogatok módszere. A véges differencia módszer előnye a könnyű programozhatóság, az utóbbi kettő – többek között – a komplex tartományok kezelése és a vegyes peremfeltételek egyszerűbb előírása. A dolgozatban a kevésbé elterjedt spektrál módszereken belül a még ritkábban alkalmazott örvényvektor-áramfüggvény formalizmusra épülő Chebyshev spektrál kollokációs módszerrel foglalkozunk. Egyszerű tartomány és megfelelően sima megoldás esetén a spektrál módszerek a magas rendű közelítés miatt különösen előnyösek a folyadékmechanikában, mivel minimális memória-és számítási igénnyel rendelkeznek a fenti három eljáráshoz képest. Számítógépes megvalósításuk azonban nehezebb, mint a három klasszikus módszeré. A dolgozat fő célja, hogy az összenyomhatatlan, időfüggő Navier-Stokes és hőátadási egyenletek megoldására általunk írt MATLAB programot adjunk és bemutassuk annak alkalmazhatóságát konkrét feladatok esetén.

A dolgozat hat fő fejezetre tagolódik. Először ismertetjük a Navier-Stokes egyenletek primitív változós alakját, majd rátérünk a számunkra fontos örvényvektor-áramfüggvény megfogalmazására. Megmutatjuk, hogy az egyes alakoknak melyek az előnyei és a hátrányai. A Navier-Stokes egyenletet kiegészítjük az energiaegyenlettel, és ily módon szabad, kényszer vagy vegyes konvekciót is tudunk számolni. A második fejezetben áttekintjük a spektrál módszereket és részletesen bemutatjuk a kollokációs spektrál módszer alkalmazását kétváltozós függvény közelítésére. A harmadik részben az első fejezetben kitűzött kezdeti-peremérték-feladatot először időben diszkretizáljuk szemi-implicit módszerrel, majd térben spektrál módszerrel és egy hatékony direkt Helmholtz-megoldót ismertetünk. Kitérünk arra, hogy mikor állítsuk le az iterációt és hogyan lehet a diszkretizációs hibát becsülni. Az ez utáni fejezet a számítógépes megvalósítással foglalkozik. Azt ezt követő fejezetben a programokat különböző feladatokon teszteljük. Egyik ilyen feladat a „lid-driven cavity” és annak regularizált változata. Az utolsó részben összefoglaljuk a következtetéseinket.

3. Alapegyenletek

Összenyomhatatlan, newtoni folyadék mozgását a Navier-Stokes és a kontinuitási egyenlet írja le. Időtől függő esetben, dimenziómentesítés után a következő alapegyenleteket kapjuk [15]:

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla) \mathbf{v} + \nabla p = \frac{1}{\text{Re}} \Delta \mathbf{v} + \mathbf{f}, \quad \mathbf{x} \in \Omega \subset \mathbb{R}^3, \quad t \in [0, T] \quad (3.1)$$

$$\mathbf{v} \cdot \nabla = 0, \quad \mathbf{x} \in \Omega \subset \mathbb{R}^3, \quad t \in [0, T]. \quad (3.2)$$

Az egyenletekben $\mathbf{x}$ a pillanatnyi konfiguráció helyvektora, $\text{Re} = vL/\nu$ a Reynolds-szám, v és L a karakterisztikus sebesség illetve hossz, $\nu = \mu/\rho_0$ a kinematikai viszkozitás, μ a dinamikai viszkozitás, ρ_0 a sűrűség. A p a nyomás és az állandó ρ_0 hányadosa.

A (3.1)-(3.2) egyenletek megoldására nincs olyan módszer, amely minden körülmény között legelőnyösebb lenne. A megoldást nehezíti, hogy még lineáris anyagegyenletet feltételezve is a (3.1) konvektív tagja miatt a differenciálegyenlet nemlineáris. A fő megoldási nehézség azonban az, hogy a kontinuitási egyenletben nem jelenik meg a nyomás, így mind végeselemes, mind spektrál diszkretizáláskor az együtthatómátrix egy zérusmátrix blokkot tartalmaz, ami az együtthatómátrixot szingulárisá teszi [15]. Ennek okára és kiküszöbölésére a 3.1. alfejezetben térünk ki. Ha $\text{Re} \ll 1$, akkor a konvektív erőkkel szemben a viszkózus erők (diffúzió) dominálnak és így a $(\mathbf{v} \cdot \nabla) \mathbf{v}$ tag elhagyható: ekkor kapjuk meg a Stokes egyenletet.

A (3.1)-(3.2) megoldását a

$$\mathbf{v} = \mathbf{v}_0, \quad \mathbf{x} \in \Omega \cup \partial\Omega, \quad t = 0 \quad (3.3)$$

kezdeti feltétel és a

$$\mathbf{v} = \mathbf{v}^D, \quad \mathbf{x} \in \partial\Omega^D, \quad t \in [0, T], \quad (3.4)$$

$$\boldsymbol{\sigma} \cdot \mathbf{n} = \mathbf{f}^N, \quad \mathbf{x} \in \partial\Omega^N, \quad t \in [0, T] \quad (3.5)$$

peremfeltételek mellett keressük, ahol $\boldsymbol{\sigma}$ a Cauchy-féle feszültségtenzor, $\mathbf{n}$ pedig a felület kifelé mutató normálisa. A kezdeti sebességmező kompatibilis, ha $\nabla \cdot \mathbf{v}_0 = 0$ és $\mathbf{v}_0 \cdot \mathbf{n} = \mathbf{v}^D \cdot \mathbf{n}$. A nyomáshoz nincs kezdeti és peremfeltételünk, de nem is szükséges, mert a (3.1)-ben a p gradiense szerepel. Emiatt a nyomás csak egy additív konstans erejéig meghatározott. Hogy a megoldás egyértelmű legyen a nyomás átlagértékét 0-nak választjuk:

$$\int_{\Omega} p \, dV = 0,$$

és Lagrange-multiplikátorral a megoldandó egyenletek közé vesszük.

A következő alpontokban a Navier-Stokes egyenletnek az alapváltozók szerinti átfogalmazását mutatjuk be.

3.1. Primitív változós formalizmus

A (3.1)-(3.2) differenciálegyenleteket primitív változós egyenleteknek nevezik, mert a $\mathbf{v}$ és p változókra kell megoldani, azokra az alapváltozókra, amire a modellt származtatták. A (3.1)-(3.2) egyenletek esetén felmerül a probléma, hogy ha a nyomást és a sebességet ugyanakkora fokszámú polinomokkal közelítjük (akár végeselem módszerrel, akár spektrál diszkretizációval), a nyomás nem határozható meg egyértelműen. Bizonyított, hogy ez a sebességmező meghatározását nem befolyásolja. A végeselem módszerben ezért olyan elemeket használnak (gyakran a Taylor-Hood elemcsaládot), amely eleget tesz az LBB (Ladyzhenskaya-Babuška-Brezzi)-féle inf-sup feltételnek. Ha a nyomást eggyel alacsonyabb fokszámú polinommal közelítjük, mint a sebességet, akkor ezt $\mathbb{P}_N - \mathbb{P}_{N-1}$ közelítésnek nevezik. A spektrálem módszereknél $\mathbb{P}_N - \mathbb{P}_{N-2}$ közelítést alkalmaznak [9]. Másik lehetőség, hogy bár a nyomást és a sebességet ugyanakkora fokszámmal közelítjük, de különböző csomópontokban keressük: a sebesség koordinátákat a Legendre-Gauss-Lobatto vagy Chebyshev-Gauss-Lobatto pontokban, a nyomást a Legendre-Gauss vagy Chebyshev-Gauss pontokban. Ezt eltolt rácsos („staggered grid”) módszernek nevezzük. Az eltolt rács programozása főleg inhomogén peremfeltételek esetén nehéz [20]. Azt is megtehetjük, hogy csak egy rácsot használunk (Lobatto integrálási pontok tenzorszorzata), de a nyomást a belső pontokban vesszük. További lehetőség, hogy növeljük a kollokációs pontok számát és így egy túlhatározott egyenletrendszert oldunk meg legkisebb négyzetes értelemben. A túlhatározottság sokszor növeli a megoldási pontosságot. A gyakran használt projekciós módszert először Chorin [5] hozta létre, melynek lényege, hogy első lépésben egy ideiglenes sebességmezőt számítunk ki, ami kielégíti a peremfeltételeket, de a (3.2) összenyomhatatlanságot nem. A második, projekciós lépésben a korábban kapott átmeneti sebességmezőt leképezzük forrásmentes sebességmezőre – innen kapta az eljárás a nevét. Ebben a lépésben a nyomásra Neumann-féle peremfeltétel van előírva, ami fizikailag mesterséges, továbbá az eljárással csökken a konvergenciasebesség a nyomásra nézve. A projekciós módszer azért előnyös, mert a nyomás és a sebesség szétválasztásával csökken az egyenletrendszer mérete és azonos fokszámú polinom közelítés használható. Weinan és Liu [23] bevezette az úgynevezett „gauge” módszert, ami kiküszöböli a nyomást, így nem kell arra mesterséges peremfeltételeket előírunk. A primitív változós formalizmusnak előnyei is vannak. A kétdimenziós áramlásra elkészített programok könnyen általánosíthatók három dimenzióra. Ebben a formalizmusban legegyszerűbb hibabecslést végezni, amely lehetővé teszi az adaptív működést. A szakirodalomban legelterjedtebben még mindig a Navier-Stokes egyenletek primitív változós felírását használják.

3.2. Örvényvektor-áramfüggvény formalizmus

Ennek a megközelítésnek a legfőbb előnye az, hogy a nyomást kiküszöböli, ezért nem szükséges eltolt rácsot alkalmazni és a sebességmező forrásmentessége is eleve teljesül. Másik előnye, hogy sok esetben (például konvektív hőátadás) az áramfüggvény többet mond a jelenségről, mint a nyomás [15]. Két dimenziós áramlásoknál háromról kettőre csökken a meg-

határozandó változók száma, ezáltal gyorsabb módszer. Hátrányai közé tartozik, hogy az örvényességre nincs peremfeltétel, míg az áramfüggvényre kettő is előírt. Mivel nehézkes három dimenziós áramlásokra alkalmazni és nehéz a peremfeltételek előírása, ezért a kereskedelmi programok a sebesség-nyomás felírásra építenek.

Értelmezzük az ω örvényvektort és ψ vektor potenciált az alábbi módon:

$$\omega = \nabla \times \mathbf{v}, \quad (3.6)$$

$$\mathbf{v} = \nabla \times \psi. \quad (3.7)$$

A dolgozatban kétdimenziós áramlásokkal foglalkozunk descartes-i koordináta rendszerben. Ekkor $\mathbf{v} = u\mathbf{i} + v\mathbf{j}$, $\psi = \psi\mathbf{k}$ és a ψ -t áramfüggvénynek nevezik, valamint a (3.7) egyenletből

$$u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}. \quad (3.8)$$

Ha a (3.7) bal oldali rotációját vesszük, akkor egy skalár egyenletet kapunk:

$$\Delta \psi + \omega = 0. \quad (3.9)$$

A (3.1) bal oldali rotációját véve az örvényvektorra vonatkozó mozgásegyenlet [18]:

$$\frac{\partial \omega}{\partial t} + \mathbf{v} \cdot \nabla \omega = \frac{1}{\text{Re}} \Delta \omega + F, \quad (3.10)$$

ahol $F = (\nabla \times \mathbf{f}) \cdot \mathbf{k}$.

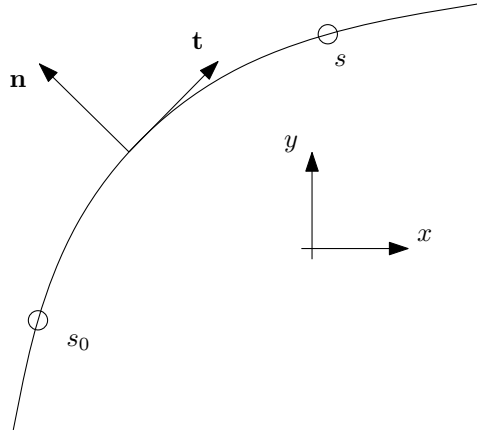
Az örvényességet és az áramfüggvényt másodlagos változóknak nevezik. Az áramfüggvény ismeretében (3.8)-cal megkaphatók a sebességszámok, a nyomást egy Poisson egyenlet Neumann-feladatának megoldásával kapjuk [18]:

$$\begin{aligned} \Delta p &= \nabla \cdot \left[\mathbf{f} - (\mathbf{v} \cdot \nabla) \mathbf{v} \right], \quad \mathbf{x} \in \Omega, \\ \frac{\partial p}{\partial n} &= \mathbf{f} \cdot \mathbf{n} - (\mathbf{v} \cdot \nabla) \mathbf{v} \cdot \mathbf{n} + \frac{1}{\text{Re}} \Delta \mathbf{v} \cdot \mathbf{n} - \frac{\partial \mathbf{v}}{\partial t} \cdot \mathbf{n} \quad \mathbf{x} \in \partial \Omega. \end{aligned} \quad (3.11)$$

Tekintsük a (3.4) Dirichlet-féle peremfeltételeket. Az áramfüggvény (3.8) definíciója alapján $\nabla \psi = (-v, u)$. Vegyük fel a peremgörbe t érintő irányú egységvektorát úgy, hogy az áramutató járásával megegyező körüljárási irányt mutassa (1. ábra); az $\mathbf{n}$ normálist kifelé irányítjuk. Legyen az érintő egységvektor $\mathbf{t} = (t_1, t_2)$ az $x - y$ koordináta-rendszerben. Az irányítás miatt így $\mathbf{n} = (-t_2, t_1)$. Ezért a peremen

$$\nabla \psi \cdot \mathbf{t} = -\mathbf{v}^D \cdot \mathbf{n}, \quad (3.12)$$

$$\nabla \psi \cdot \mathbf{n} = \mathbf{v}^D \cdot \mathbf{t}. \quad (3.13)$$



1. ábra. Az érintő és normális irányítása a peremen

Vegyük a (3.12) vonalintegrálját egy rögzített $s_0(x, y)$ és egy változó $s(x, y)$ határ között:

$$\int_{s_0}^s \nabla \psi \cdot \mathbf{t} \, d\hat{s} = \psi(s) - \psi(s_0) = - \int_{s_0}^s \mathbf{v}^D \cdot \mathbf{n} \, d\hat{s}.$$

Mivel a (3.8), (3.9), (3.10) egyenletekben a ψ deriváltja szerepel csak, ezért az áramfüggvényt csak egy időtől függő függvény erejéig lehet meghatározni, így feltehetjük, hogy $\psi(s_0) = 0$ és így

$$\psi = - \int_{s_0}^s \mathbf{v}^D \cdot \mathbf{n} \, d\hat{s}. \quad (3.14)$$

A (3.13) egyenlet egy másik feltételt ad:

$$\frac{\partial \psi}{\partial n} = \mathbf{v}^D \cdot \mathbf{t}. \quad (3.15)$$

A kezdeti örvényesség a kezdeti sebességből számítható, az áramfüggvény kezdeti értékére nincs szükség, mert nem szerepel a differenciálegyenletekben. A Navier-Stokes egyenlet örvényvektor-áramfüggvény megfogalmazásában nehézséget jelent, hogy az örvényességre nincs peremfeltétel, míg az áramfüggvényre két peremfeltétel is elő van írva.

Tehát ha a sebességre Dirichlet-féle peremfeltételek vannak előírva, akkor a (3.9)-(3.10) differenciálegyenleteket kell megoldani a (3.14)-(3.15) peremfeltételekkel.

3.3. Egyéb formalizmusok

Az előző módszer hátránya, hogy a peremfeltételek nehezen kezelhetők. Ezt kiküszöböli a tisztán áramfüggvény megfogalmazás. Ha (3.9)-et a (3.10)-be helyettesítjük, ψ -re a biharmonikus egyenlet első peremérték-feladatát kapjuk. Ennek előnye, hogy csak egy ismeretlen szerepel, viszont spektrális közelítésnél a magasabb rendű deriváló mátrixok nagyobb kerekítési hibával hozhatók létre, míg véges elem módszernél magasabb rendű közelítést kell alkalmazni. Spektrál módszereknél további probléma a (3.15) megadása. Schultz és társai [19]-ban egy túl-

határozott egyenletrendszer megoldásával adták meg a feladat megoldását, illetve Karageorghis [12] perempontokon kívüli kollokációs pontokat is felhasznált, hogy a (3.15)-öt előírja.

Létezik sebesség-örvényvektor formalizmus, amelynek előnye, hogy a nyomás hiánya miatt könnyebb előírni a peremfeltételt félig végtelen tartományon és az örvénylő áramlásokat is jól kezeli (l. [8]).

Van ezen kívül örvényvektor-sebesség-nyomás módszer, továbbá ez előző módszerek módosításai.

3.4. Konvektív hőátadás

A hőmérsékletmező meghatározásához a Boussinesq-közelítést alkalmazzuk. Ez a Navier-Stokes egyenletek és a termodinamika I. főtételének bizonyos feltételekkel való egyszerűsítésével kapható, melyek a következők [18]:

- A (3.1) mozgásegyenlet dimenzióval rendelkező alakjában a ρ sűrűség állandó, kivéve az $\mathbf{f}$ felhajtóerőt:

$$\mathbf{f} = \frac{\rho}{\rho_0} \mathbf{g},$$

ahol ρ_0 az állandó referencia nyomás, $\mathbf{g}$ pedig a nehézségi gyorsulás.

- A kontinuitási egyenlet megmarad a (3.2) alakban, tehát ott a sűrűséget állandónak vesszük.
- Az energiaegyenlet az alábbi alakot ölti:

$$\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T - \kappa \Delta T = 0, \quad (3.16)$$

ahol T a hőmérséklet, κ a hőmérséklet-vezetési tényező.

A nyomás és a hőmérséklet közötti kapcsolatot ekkor lineáris állapotegyenlet írja le:

$$\rho = \rho_0[1 - \beta(T - T_0)],$$

ahol T_0 a ρ_0 -hoz tartozó referencia hőmérséklet, β a térfogati hőtágulási együttható. Kis hőmérséklet különbségek esetén a Boussinesq-közelítés reális eredményt ad.

Örvényvektor-áramfüggvény változóiban dimenziómentes alakban a kapcsolt parciális differenciálegyenlet-rendszer a Boussinesq-közelítéssel élve a

$$\begin{aligned} \Delta \psi + \omega &= 0, \\ \frac{\partial \omega}{\partial t} + \mathbf{v} \cdot \nabla \omega &= \frac{1}{\text{Re}} \Delta \omega + \text{Ri} \frac{\partial T}{\partial x}, \\ \frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T &= \frac{1}{\text{RePr}} \Delta T \end{aligned} \quad (3.17)$$

alakban írható fel, ahol $\text{Ri} = g\beta$ a Richardson-szám, $\text{Pr} = \nu/\kappa$ a Prandtl-szám [16]. A hőmérsékletre előírhatjuk a

$$T = T_b$$

Dirichlet-peremfeltételt, azaz hőmérséklet-eloszlást a peremen, vagy a

$$\frac{\partial T}{\partial n} = T_f$$

Neumann-feltételt, vagyis megadott hőfluxust. Adott továbbá a $T(\mathbf{x}, t = 0) = T_0(\mathbf{x})$ kezdeti feltétel.

Az 5. fejezetben ezen egyenletek spektrál módszerrel való megoldásával foglalkozunk.

4. Spektrál módszerek

A spektrál módszereket differenciálegyenletek megoldására az 1970-es évek elejétől alkalmazzák [4]. Leginkább a folyadékmechanikában és a meteorológiában terjedtek el, ahol magas rendű közelítések szükségesek. A spektrál módszereknek mindhárom típusa (a kollokációs, a Galerkin és a tau spektrál módszerek) a súlyozott maradékok módszeréhez tartozik és fő osztályozásuk a próbafüggvények típusa szerint történik. A próbafüggvények a Galerkin módszernél kielégítik a peremfeltételeket és megegyeznek a súlyfüggvényekkel, kollokációs módszernél a súlyfüggvény az argumentumában eltolt Dirac-delta függvény. Míg a kollokációs módszernél a differenciálegyenletnek bizonyos pontokban – az úgynevezett kollokációs pontokban – kell fennállnia, addig a Galerkin módszernél a maradéknak a súlyfüggvényekkel vett szorzata integráljának kell nullának lennie. A tau módszer a Galerkin módszerhez hasonlóan írja elő a differenciálegyenlet teljesülését, viszont a tesztfüggvényeknek itt nem kell kielégíteniük a peremfeltételeket, ezért plusz egyenletek kellenek a peremfeltételek teljesüléséhez [14]. A Galerkin és tau spektrál módszerek a sorfejtést felhasználva a csonkított sor együtthatóira adnak megoldást, míg kollokációnál interpolációt alkalmazva az interpolációs polinomnak a kollokációs pontokban (csomópontokban) felvett függvényértékeit határozzuk meg. Az első módszer másik neve a modális közelítés, az utóbbi csomóponti. A sorfejtési együtthatók ismeretében meghatározhatók a csomóponti értékek és fordítva is. Kollokációs módszereknél legjobb közelítést akkor kapunk, ha a kollokációs pontok egybeesnek numerikus integrálások alappontjaival (l. Chebyshev-Gauss-Lobatto pontok, 4.1. fejezet). Periodikus peremfeltételeknél a Fourier-módszert, nem periodikus, $[-1, 1]^d$ véges tartomány esetén a Jacobi-polinomokat (leggyakrabban a Chebyshev és a Legendre polinomokat) használjuk. A $[0, \infty)^d$ tartománynál Laguerre, a $(-\infty, \infty)^d$ tartománynál Hermite polinomok a tesztfüggvények. Nem véges intervallum kezelésének további módjai: csonkítás véges tartományra, vagy leképezések bevezetésével $[-1, 1]^d$ -re transzformáljuk.

4.1. Interpoláció

A $T_n(x)$ elsőfajú n -edfokú Chebyshev polinomokat az $x \in [-1, 1]$ intervallumon a

$$T_n(x) = \cos(n \arccos x), \quad n = 0, 1, \dots \quad (4.1)$$

összefüggés értelmezi, amely kifejezhető a

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), \quad n > 1$$

háromtagú rekurzióval is. A $\{T_k(x)\}_{k=0}^N$ ortogonális függvényrendszert alkot a $w(x) = (1 - x^2)^{-1/2}$ súlyfüggvénnyel, azaz

$$\langle T_k, T_l \rangle_w = \int_{-1}^1 T_k T_l w \, dx = \frac{\pi}{2} c_k \delta_{kl},$$

ahol

$$c_k = \begin{cases} 2 & \text{ha } k = 0, \\ 1 & \text{ha } k = 1. \end{cases}$$

Az elsőfajú Chebyshev polinomok szélsőértékhelyei egybeesnek a Gauss-Lobatto integrálási pontokkal, így azokat Chebyshev-Gauss-Lobatto pontoknak nevezzük és a következők:

$$x_i = \cos \frac{i\pi}{N}. \quad (4.2)$$

Az $u(x)$ függvényt a Chebyshev-sor N -edik szeletével közelítve:

$$I_N u(x) = \sum_{k=0}^N a_k T_k(x). \quad (4.3)$$

A fenti közelítést spektrál bázisban végeztük el. A spektrál polinom interpoláció csomóponti alakban is megadható Lagrange bázisban:

$$p_N u(x) = \sum_{j=0}^N u_j \ell_j(x), \quad (4.4)$$

ahol

$$\ell_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^N \frac{x - x_k}{x_j - x_k}, \quad \ell_j(x_k) = \delta_{kj}. \quad (4.5)$$

A baricentrikus Lagrange-interpoláció gyorsaságáról és numerikus stabilitásáról ismert [2] és a

$$p_N(x) = \frac{\sum_{j=0}^N \frac{w_j}{x - x_j} u_j}{\sum_{j=0}^N \frac{w_j}{x - x_j}} \quad (4.6)$$

alakban áll elő, ahol u_j ($j = 0, \dots, n$) az x_j pontokban ismert függvényértékek és w_j az x_j pontokhoz tartozó baricentrikus súlyok a

$$w_j = \frac{1}{n \prod_{\substack{k=0 \\ k \neq j}} (x_j - x_k)}, \quad j = 0, \dots, n \quad (4.7)$$

módon számítva.

Kétváltozós függvény közelítésekor mindkét irányban Chebyshev polinomokat alkalmazunk és az interpolációs pontokból tenzorszorzat rácsot alkotunk. Az $u(x, y)$ függvény tenzor-

szorzat közelítése spektrális bázisban

$$I_N(x, y) = \sum_{i=0}^N \sum_{j=0}^N a_{i,j} T_i(x) T_j(y), \quad (4.8)$$

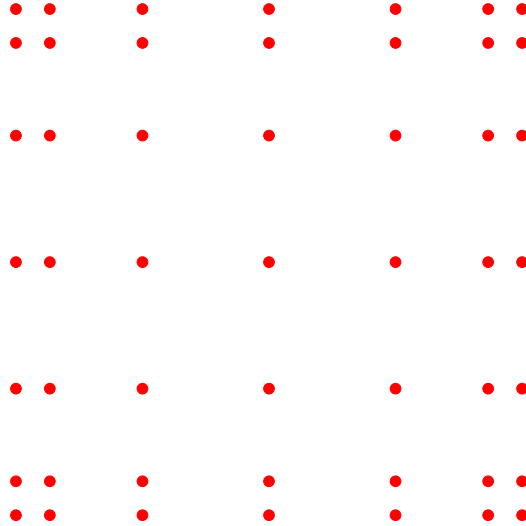
csomóponti bázisban pedig

$$p_N(x, y) = \sum_{i=0}^N \sum_{j=0}^N u_{i,j} \ell_i(x) \ell_j(y). \quad (4.9)$$

Az egyszerűbb jelölés miatt a továbbiakban ha két index van a szummában, akkor azon kettős összegzést értünk:

$$\sum_{n,m=0}^{N,M} \equiv \sum_{n=0}^N \sum_{m=0}^M.$$

A 2. ábrán a (4.2) pontokból álló rács látható $N = M = 6$ esetén.



2. ábra. Elsőfajú hatodfokú Chebyshev-polinomból alkotott tenzorszorzat rács

Spektrál módszerekkel nagy pontosság érhető el viszonylag kevés számú interpolációs alap-ponttal. Grafikus ábrázolásnál így szükség lehet több pontban ismerni a közelítő függvény-értékeket. Tenzorszorzat rácsnál megtehető, hogy a függvényértékeket interpoláljuk az egyik koordináta irányban, majd az így kapott interpolációs együtthatókat a másik irányban [13]:

$$\begin{aligned} \bar{U}_{n,j} &= \sum_{i=0}^{N^{\text{régi}}} T_{ni}^{(x)} u_{i,j}, \quad n = 0, \dots, N^{\text{új}}, \quad j = 0, \dots, M^{\text{régi}}, \\ U_{n,m} &= \sum_{j=0}^{M^{\text{régi}}} T_{mj}^{(y)} \bar{U}_{n,j}, \quad n = 0, \dots, N^{\text{új}}, \quad m = 0, \dots, M^{\text{új}}, \end{aligned} \quad (4.10)$$

ahol $\mathbf{T}^{(x)}$ és $\mathbf{T}^{(y)}$ (4.6) alapján számítható.

4.2. Spektrális deriválás

A spektrális deriválásnak három elterjedt változata van. Megtehetjük, hogy a deriváltat a spektrál bázisban értékeljük ki. Ekkor minden tartománybeli pontban rendelkezésünkre áll a derivált közelítő értéke. Elvégezhetjük a deriválást a fizikai térben is vagy mátrixszorzással, vagy pedig transzformációs módszerekkel. A fizikai térben való deriválás során – hasonlóan az interpolációhoz – nem a Chebyshev-együtthatókra, hanem a csomópontbeli értékekre kapunk megoldást. A mátrixszorzásos módszer egyszerűen megvalósítható és párhuzamosítható, viszont nagyobb foksám esetén a transzformációs módszerek gyorsabbak. Fourier és Chebyshev módszereknél alkalmazható a gyors Fourier-transzformáció (Fast Fourier Transform, FFT).

Egyváltozós függvény deriváltja az interpolációs pontokban

$$u'(x_i) \approx (p_N(u(x_i)))' = \sum_{j=0}^N u_j \ell_j'(x_i) = \sum_{j=0}^N D_{ij} u_j, \quad i = 0, \dots, N \quad (4.11)$$

módon képezhető, ahol D az első deriváló mátrix, melyre

$$D_{ij} = \begin{cases} \frac{w_j}{w_i} \left(\frac{1}{x_i - x_j} \right), & i \neq j, \\ -\sum_{\substack{j=0 \\ j \neq i}}^N D_{ij}, & i = j. \end{cases} \quad (4.12)$$

A p -edrendű derivált közelítése:

$$\frac{d^p u(x_i)}{dx^p} \approx (p_N(u(x_i)))^{(p)} = \sum_{j=0}^N u_j \ell_j^{(p)}(x_i) = \sum_{j=0}^N D_{ij}^{(p)} u_j, \quad i = 0, \dots, N. \quad (4.13)$$

A $D^{(p)}$ p -edik deriváló mátrixot rekurzióval számíthatjuk a következőképpen:

$$D_{ij}^{(p)} = \begin{cases} \frac{p}{x_i - x_j} \left(\frac{w_j}{w_i} D_{ii}^{(p-1)} - D_{ij}^{(p-1)} \right), & i \neq j, \\ -\sum_{\substack{j=0 \\ j \neq i}}^N D_{ij}^{(p)}, & i = j. \end{cases} \quad (4.14)$$

Megjegyzés: A (4.12), (4.14) összefüggések nem csak Chebyshev-polinomokra alkalmazhatók. Ha a differenciálegyenlet gyenge alakjával dolgozunk, akkor célszerű Legendre-polinomokat használnunk, hiszen ott a súlyfüggvény azonosan 1. Tehát az $\{x_i\}_{i=0}^N$ interpolációs alappontok ismeretében a baricentrikus súlyok (4.7) alapján, a deriváló mátrixok pedig (4.12) - (4.14)-ből számíthatók.

Kétváltozós függvény deriváltjai a tenzorszorzat interpoláció miatt egyszerűen meghatá-

rozhatók (mindkét koordináta irányban azonos, N fokszámú polinommal közelítve):

$$\frac{\partial u(x_i, y_j)}{\partial x} = \sum_{k,l=0}^N u_{k,l} \ell'_k(x_i) \ell_l(y_j) = \sum_{k,l=0}^N u_{k,l} D_{ik} \delta_{jl} = \sum_{k=0}^N u_{k,j} D_{ik} = \mathbf{DU}, \quad (4.15)$$

$$\frac{\partial u(x_i, y_j)}{\partial y} = \sum_{k,l=0}^N u_{k,l} \ell_k(x_i) \ell'_l(y_j) = \sum_{k,l=0}^N u_{k,l} D_{jl} \delta_{ik} = \sum_{l=0}^N u_{i,l} D_{jl} = \mathbf{UD}^T. \quad (4.16)$$

4.3. A spektrál módszerek jellemzői

A véges differencia, végeselem, véges térfogatok módszerével szemben a spektrál módszerek globális módszerek, azaz a közelítő megoldást a teljes tartományon állítják elő. Ez előnyös a pontosság szempontjából, viszont a peremfeltételekre különösen figyelni kell. Ugyanis egy sor belső pontjai kapcsolatban állnak az adott sor többi pontjával – így a perempontokkal – is. Tehát a peremfeltételek közvetlen hatással vannak a megoldásra. A globális jelleg másik következménye, hogy a lokális szingularitások miatt a szingularitásoktól távol is csökken a megoldás pontossága [18]. Éppen ezért a spektrál módszerek megfelelően sima megoldások esetén hatékonyak. Léteznek egyéb módszerek (tartományfelosztási módszerek, szingularitás eltávolítása), amelyekkel ez a hátrány kiküszöbölhető. A spektrálem-módszer alkalmas komplex tartományok kezelésére. A Chebyshev közelítés során kapott mátrixok rosszul kondicionált, nem szimmetrikus, indefinit, telt mátrixok. Emiatt kis N esetén direkt eljárással oldjuk meg a lineáris egyenletrendszert. Nagyobb N esetén a nagy tárhely és számítási idő miatt iteratív módszerrel kell dolgozni és a nagy kondíciós szám miatt mindenképpen kell használni prekon-dicionálót. A prekon-dicionáló épülhet véges differencia, vagy pedig végeselem módszerre. A spektrális közelítés hibája $\mathcal{O}(N^{-m})$, ahol m a folytonos deriváltak száma. Ha m minden határon túl nő, akkor exponenciális konvergenciát kapunk, szemben a végeselem módszer algebrai konvergenciájával. További előnye a spektrál módszereknek, hogy a hiba a spektrális együtthetők csökkenéséből becsülhető.

4.4. Általánosítás négyszög tartományra

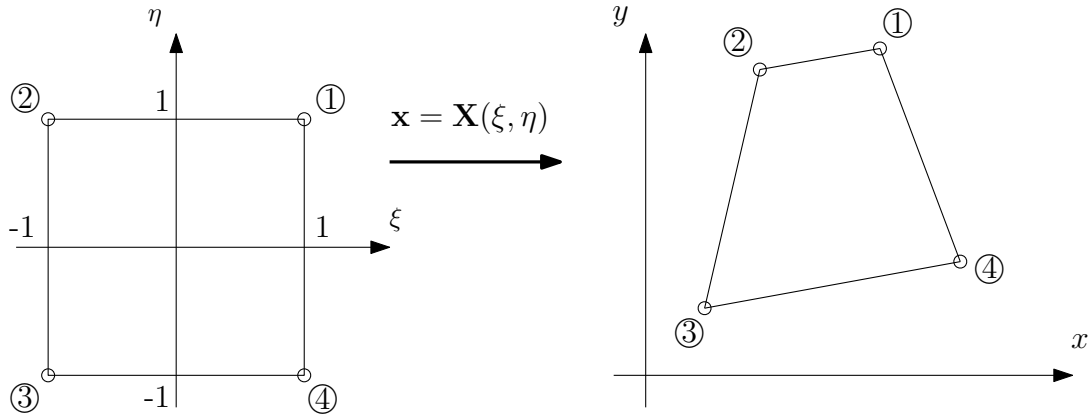
Korábban a $[-1, 1]^2$ tartományon dolgoztunk. Az $\mathbf{x} = (x, y) = \mathbf{X}(\xi, \eta)$ leképezéssel kapcsolat hozható létre az (x, y) fizikai és a (ξ, η) számítási tartomány között. Görbe vonalú alakzatokkal nem foglalkozunk, csak olyan transzformációval, ami a két egység oldalhosszú négyzetet tetszőleges négyszögbe viszi át.

A 3. ábrán látható leképezést az alábbi bilineáris alakfüggvények valósítják meg:

$$\begin{aligned} N_1(\xi, \eta) &= \frac{1}{4}(1 + \xi)(1 + \eta), & N_2(\xi, \eta) &= \frac{1}{4}(1 - \xi)(1 + \eta), \\ N_3(\xi, \eta) &= \frac{1}{4}(1 + \xi)(1 - \eta), & N_4(\xi, \eta) &= \frac{1}{4}(1 - \xi)(1 - \eta). \end{aligned} \quad (4.17)$$

Így a leképezés a következő:

$$x(\xi, \eta) = \sum_{i=1}^4 N_i(\xi, \eta)x_i, \quad y(\xi, \eta) = \sum_{i=1}^4 N_i(\xi, \eta)y_i \quad (4.18)$$



3. ábra. A referencia tartomány leképezése négyszög tartományra

Nemcsak a független, hanem a függő változók is változnak a transzformáció hatására. Az új x és y koordináta-tengelyek mentén az u függvény parciális deriváltai:

$$\begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \end{bmatrix}.$$

Általános esetben a ξ és η deriváltjait nem tudjuk meghatározni. Tegyük fel, hogy az $\mathbf{X}^{-1} : (x, y) \rightarrow (\xi, \eta)$ inverz leképezés létezik. Ekkor

$$\begin{bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix}.$$

Az inverz leképezés definíciója miatt így

$$\begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix}^{-1} = \mathbf{J}^{-1}. \quad (4.19)$$

A leképezés nem elfajuló, ha $\det \mathbf{J} \neq 0$.

Jelölje (x_i, y_i) , $i = 1, \dots, 4$ a 3. ábrán lévő jobb oldali négyszög csúcsainak koordinátáit.

Felhasználva (4.18)-at, a Jacobi-mátrix inverze a következő:

$$\mathbf{J}^{-1} = \frac{2}{A\xi + B\eta + C} \begin{bmatrix} a\xi + b & -a\eta + c \\ d\xi + e & -d\eta + f \end{bmatrix}, \quad (4.20)$$

ahol

$$\begin{aligned} A &= (x_1 - x_2)(y_3 - y_4) + (x_4 - x_3)(y_1 - y_2), & B &= (x_3 - x_2)(y_1 - y_4) + (x_1 - x_4)(y_2 - y_3), \\ C &= (x_1 - x_3)(y_2 - y_4) + (x_2 - x_4)(y_3 - y_1), \\ a &= y_1 - y_2 + y_3 - y_4, & b &= y_1 + y_2 - y_3 - y_4, \\ c &= -y_1 + y_2 + y_3 - y_4, & d &= -x_1 + x_2 - x_3 + x_4, \\ e &= -x_1 - x_2 + x_3 + x_4, & f &= x_1 - x_2 - x_3 + x_4. \end{aligned}$$

Ha téglalap tartományról van szó (azaz $x_4 = x_1, x_3 = x_2, y_2 = y_1, y_4 = y_3$), akkor

$$\mathbf{J}^{-1} = \begin{bmatrix} \frac{2}{x_1 - x_2} & 0 \\ 0 & \frac{2}{y_1 - y_3} \end{bmatrix}. \quad (4.21)$$

5. A kezdeti-peremértékfeladat numerikus megoldása

5.1. A Helmholtz-egyenlet megoldása Dirichlet-peremfeltételekkel

A következőkben a $\Delta u - \sigma u = f$ Helmholtz-egyenletet oldjuk meg Descartes-féle síkbeli koordináta rendszerben Dirichlet-peremfeltételekkel. Sok mechanikai, hőtani, elektromágneses jelenség matematikai modellje a Helmholtz-egyenletre vezet. Az 5.3. részben a Navier-Stokes egyenlet időbeli diszkretizációja során minden időpillanatban egy Helmholtz-egyenletet kell megoldanunk. A megoldandó peremérték-feladat tehát

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - \sigma u = f, \quad (x, y) \in \Omega \quad (5.1)$$

$$u = g, \quad (x, y) \in \Gamma = \partial\Omega, \quad (5.2)$$

ahol Ω a $(-1, 1) \times (-1, 1)$ négyzet tartomány, Γ pedig a pereme. Az (5.2) a négy oldalon az alábbi módon írható:

$$\begin{aligned} u(-1, y) &= g_l(y), & -1 \leq y \leq 1 \\ u(x, 1) &= g_t(x), & -1 \leq x \leq 1 \\ u(1, y) &= g_r(y), & -1 \leq y \leq 1 \\ u(x, -1) &= g_b(x), & -1 \leq x \leq 1. \end{aligned} \quad (5.3)$$

Kollokációs pontoknak az

$$(x_i, y_j) = \left(-\cos \frac{i\pi}{N}, -\cos \frac{j\pi}{N} \right), \quad i, j = 0, \dots, N \quad (5.4)$$

pontokat választjuk, a pontok számozását a (4.2)-höz képest programozás technikai okok miatt megcseréltük. Az $u(x, y)$ függvényt a (4.9) szerint közelítjük, a parciális deriváltakat pedig a (4.15)-(4.16)-hoz hasonlóan. Így az (5.1) diszkrét alakja:

$$\begin{aligned} \sum_{k=0}^N u_{k,j} D_{ik}^{(2)} + \sum_{k=0}^N u_{i,k} D_{jk}^{(2)} - \sigma u_{i,j} &= f_{i,j}, \quad i, j = 1, \dots, N-1 \\ \sum_{k=1}^{N-1} D_{ik}^{(2)} u_{k,j} + \sum_{k=1}^{N-1} u_{i,k} D_{jk}^{(2)} - \sigma u_{i,j} &= f_{i,j} - D_{i0}^{(2)} u_{0j} - D_{iN}^{(2)} u_{Nj} - D_{j0}^{(2)} u_{i0} - D_{jN}^{(2)} u_{iN}. \end{aligned} \quad (5.5)$$

Az (5.5) jobb oldalán felhasználjuk az (5.3) peremfeltételeket:

$$\sum_{k=1}^{N-1} D_{ik}^{(2)} u_{k,j} + \sum_{k=1}^{N-1} u_{i,k} D_{jk}^{(2)} - \sigma u_{i,j} = f_{i,j} - D_{i0}^{(2)} g_b(x_j) - D_{iN}^{(2)} g_t(x_j) - D_{j0}^{(2)} g_l(y_i) - D_{jN}^{(2)} g_r(y_i), \quad (5.6)$$

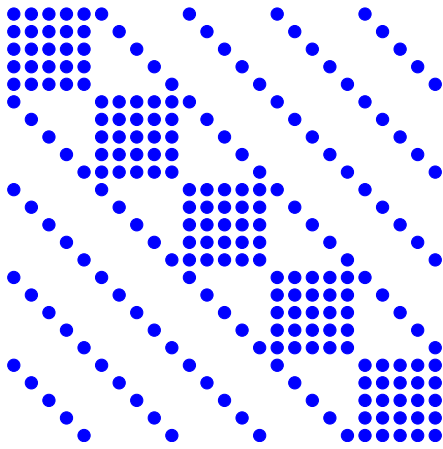
azaz mátrixos alakban:

$$\mathbf{D}^{(2)} \mathbf{U} + \mathbf{U} \mathbf{D}^{(2)T} - \sigma \mathbf{U} = \mathbf{H}. \quad (5.7)$$

Az (5.7) lineáris mátrixegyenlet megoldható direkt, vagy iteratív eljárással. A direkt megoldáshoz általános esetben az $u_{i,j}$ két indexes ismeretleneket egy oszlopvektorba kell elrendezni, hogy lineáris egyenletrendszerként meg tudjuk oldani. Mivel a MATLAB – a Fortranhoz hasonlóan – oszlopfolytonosan tárolja a tömböket, ezért az $n = \text{index}(i, j) = i + (j - 1)(N - 1)$ transzformációt alkalmazzuk. Az $(N - 1)^2$ belső pont miatt az együtthatómátrix mérete $(N - 1)^2 \times (N - 1)^2$ és felírható a Kronecker-féle tenzorszorzattal [10]:

$$(\mathbf{I} \otimes \mathbf{D}^{(2)} + \mathbf{D}^{(2)} \otimes \mathbf{I} - \sigma \mathbf{I} \otimes \mathbf{I}) \mathbf{u} = \mathbf{g}, \quad (5.8)$$

ahol $\mathbf{u}$ az $u_{i,j}$ értékeket, míg $\mathbf{g}$ az (5.6) jobb oldalát tartalmazó oszlop mátrix. Az együtthatómátrix nem-nulla elemeinek elhelyezkedése a 4. ábrán látható $M = N = 6$ közelítésre.



$M = N$	$\frac{\text{nem-nulla elemek száma}}{\text{összes elem száma}}$
5	0,4375
10	0,2099
20	0,1025
30	0,0678
50	0,0404
75	0,0268

2. táblázat. Az együttható mátrix ritkasága

4. ábra. A Helmholtz-operátor spektrál kollokációval diszkrétizált Kronecker szorzata

A 2. táblázat a nem-nulla elemek és az összes elem számának hányadosát tartalmazza. Látható, hogy N növelésével az együtthatómátrix ritka mátrix, de nem olyan mértékben, mint véges differencia, vagy végeselemes közelítésnél.

Bizonyos differenciáloperátorok esetében – mint például a Helmholtz-operátor – részleges vagy teljes diagonalizálást alkalmazhatunk, ami jelentősen kisebb tárhely- és számítási igényű, mint a Kronecker-szorzat [18]. Ha a $\mathbf{D}^{(2)}$ mátrix sajátvektorai lineárisan függetlenek, akkor diagonalizálható:

$$\mathbf{D}^{(2)} = \mathbf{P} \mathbf{\Lambda} \mathbf{P}^{-1}, \quad (5.9)$$

ahol $\mathbf{\Lambda}$ diagonális mátrix, főátlójában a sajátértékekkel és $\mathbf{P}$ oszlopaiban a sajátvektorok állnak. Az (5.7)-et balról $\mathbf{P}^{-1}$ -zel szorozva, majd az $\tilde{\mathbf{U}} = \mathbf{P}^{-1} \mathbf{U}$ és $\tilde{\mathbf{H}} = \mathbf{P}^{-1} \mathbf{H}$ helyettesítést bevezetve a

$$\mathbf{\Lambda} \tilde{\mathbf{U}} + \tilde{\mathbf{U}} \mathbf{D}^{(2)T} - \sigma \tilde{\mathbf{U}} = \tilde{\mathbf{H}}$$

egyenletet kapjuk. Szorozzuk meg ezt jobbról $\mathbf{P}^{-T}$ -tal. Az $\hat{\mathbf{U}} = \tilde{\mathbf{U}} \mathbf{P}^{-T}$ és $\hat{\mathbf{H}} = \tilde{\mathbf{H}} \mathbf{P}^{-T}$ helyettesítés után a

$$\mathbf{\Lambda} \hat{\mathbf{U}} + \hat{\mathbf{U}} \mathbf{\Lambda} - \sigma \hat{\mathbf{U}} = \hat{\mathbf{H}} \quad (5.10)$$

nem csatolt egyenletrendszert nyerjük. Így már $\hat{\mathbf{U}}$ elemei egymástól függetlenül számíthatók:

$$\hat{u}_{i,j} = \frac{\hat{h}_{i,j}}{\lambda_i + \lambda_j - \sigma}, \quad i, j = 1, \dots, N-1. \quad (5.11)$$

Az $\hat{\mathbf{U}}$ ismeretében az $\mathbf{U}$, az $\mathbf{U} = \mathbf{P}\hat{\mathbf{U}}\mathbf{P}^T$ összefüggéssel meghatározható.

Végül egy példán (l. [13], 170. oldal) mutatjuk be a spektrál kollokációs módszer pontosságát. Oldjuk meg a következő Poisson-egyenletet az alábbi peremfeltételekkel:

$$\nabla^2 u = -8\pi^2 \cos(2\pi x) \sin(2\pi y), \quad (x, y) \in (-1, 1) \times (-1, 1),$$

$$u(x, -1) = 0, \quad -1 \leq x \leq 1$$

$$u(1, y) = \sin(2\pi y), \quad -1 \leq y \leq 1$$

$$u(x, 1) = 0, \quad -1 \leq x \leq 1$$

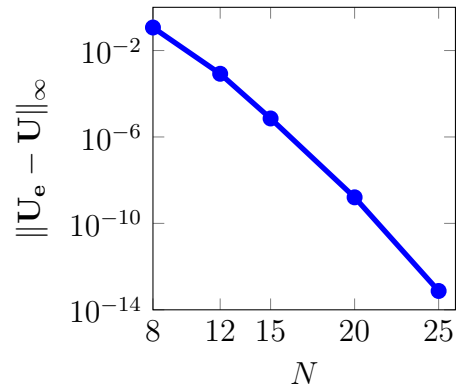
$$u(-1, y) = \sin(2\pi y) \quad -1 \leq y \leq 1.$$

Az egzakt megoldás $u(x, y) = \cos(2\pi x) \sin(2\pi y)$, pontbeli értékeiből alkotott mátrix $\mathbf{U}_e$. Az $\mathbf{U}$ közelítő megoldás ettől való maximális eltérését a kollokációs pontokban az 5. ábra mutatja. A 3. táblázat a Kronecker-szorozattal, illetve diagonalizálással való megoldás számítási idejét mutatja különböző felosztás mellett.

$N = M$	Számítási idő (s)*			
	10	30	50	70
Kronecker	0.0020	0.0619	0.5095	2.1711
diagonalizálás	0.0016	0.0029	0.0064	0.0113

3. táblázat.

* Öt futás átlaga, Intel Core i5-2500K (4,5 GHz).



5. ábra. A spektrális közelítés hibája

A 5. ábrán jól látható az exponenciális konvergencia. A 3. táblázatból látszik, hogy nagyobb N esetén a megoldás lassúvá válik. Ez különösen akkor lényeges, ha egy időtől függő feladatnál minden időlépésben meg kell oldani a Helmholtz-egyenletet. Legalább ugyanekkora probléma a lefoglalt memória. Kronecker-szorozattal az együtthatómátrix mérete $(N-1)^2 \times (N-1)^2$, és mivel dupla precíziós lebegőpontos számábrázolásnál egy szám 8 bájt helyet foglal, ezért $N = 70$ -nél a mátrix lefoglalásához 181,3 MB szükséges.

5.2. A Helmholtz-egyenlet megoldása Robin-peremfeltételekkel

Ahogy az 4. fejezetben írtuk, a Helmholtz-egyenletet megoldhatjuk kollokációs és Galerkin módszerrel is. A folytonos Galerkin módszer előnye, hogy a diszkretizálás után kapott egyenletrendszer szimmetrikus, pozitív definit mátrixú, így iteratív megoldásnál alkalmazható a könnyen kódolható, kis memóriaigényű és nagy konvergenciasebességű konjugált gradiens módszer. Mi direkt megoldást alkalmazunk diagonalizálással [18], így a kollokációs módszer is megfelelő. Megoldandó az alábbi peremérték-feladat:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - \sigma u = f, \quad (x, y) \in \Omega \quad (5.12)$$

$$\alpha u + \beta \frac{\partial u}{\partial n} = g, \quad (x, y) \in \Gamma = \partial\Omega, \quad (5.13)$$

ahol $\partial/\partial n$ a peremen a normális szerinti deriváltat jelöli, továbbá σ állandó. Az (5.3)-hoz hasonlóan az (5.13) is felírható a négy oldalon:

$$\begin{aligned} \alpha_l(y)u(-1, y) - \beta_l(y) \frac{\partial u(-1, y)}{\partial x} &= g_l(y), & -1 \leq y \leq 1 \\ \alpha_t(x)u(x, 1) + \beta_t(x) \frac{\partial u(x, 1)}{\partial y} &= g_t(x), & -1 \leq x \leq 1 \\ \alpha_r(y)u(1, y) + \beta_r(y) \frac{\partial u(1, y)}{\partial x} &= g_r(y), & -1 \leq y \leq 1 \\ \alpha_b(x)u(x, -1) - \beta_b(x) \frac{\partial u(x, -1)}{\partial y} &= g_b(x), & -1 \leq x \leq 1. \end{aligned} \quad (5.14)$$

Az egyértelmű megoldhatóság feltétele, hogy α_l , α_t , α_r és α_b egyszerre nem 0.

Ha α és β egy adott oldalon is változik, akkor a mátrix diagonalizálás nem alkalmazható közvetlenül [18]. Lehetőségünk van arra, hogy a bázisfüggvényeket módosítsuk, hogy azok eleget tegyenek a peremfeltételeknek [21], büntető paraméter segítségével is előírhatjuk a peremfeltételeket [11], vagy pedig alkalmazhatjuk az úgynevezett *influence matrix*¹ technikát. Mi ez utóbbit használjuk, amely akkor előnyös, ha az (5.12)-(5.13) feladatot sokszor kell megoldani.

A módszer alapja az, hogy kihasználjuk az (5.1) lineáris voltát, így a megoldást kereshetjük az alábbi alakban:

$$u = \tilde{u} + \bar{u}, \quad (5.15)$$

ahol $\tilde{u}$ eleget tesz a $\tilde{\mathcal{P}}$ -feladatnak:

$$\begin{aligned} \frac{\partial^2 \tilde{u}}{\partial x^2} + \frac{\partial^2 \tilde{u}}{\partial y^2} - \sigma \tilde{u} &= f, \quad (x, y) \in \Omega \\ \tilde{u} &= 0, \quad (x, y) \in \Gamma \end{aligned} \quad (5.16)$$

¹Tudomásunk szerint nincs rá magyar szó, a hatásmátrixot fogjuk használni.

míg $\bar{u}$ kielégíti a $\bar{\mathcal{P}}$ -problémát:

$$\begin{aligned} \frac{\partial^2 \bar{u}}{\partial x^2} + \frac{\partial^2 \bar{u}}{\partial y^2} - \sigma \bar{u} &= 0, \quad (x, y) \in \Omega \\ \bar{u} &= \xi, \quad (x, y) \in \Gamma. \end{aligned} \quad (5.17)$$

Az ismeretlen ξ függvényt úgy határozzuk meg, hogy az $u = \tilde{u} + \bar{u}$ kielégítse az (5.13) peremfeltételeket. Chebyshev-kollokációt használva, az (5.15)-öt N -edfokú Chebyshev polinomokkal az

$$u_N = \tilde{u}_N + \bar{u}_N \quad (5.18)$$

módon közelítjük. A $\tilde{\mathcal{P}}$ -feladat a homogén Dirichlet-peremfeltétel miatt megoldható mátrix diagonalizálással (l. 5.1. rész). A $\bar{\mathcal{P}}$ -feladat megoldását keressük az

$$\bar{u}_N(x, y) = \sum_{l=1}^L \xi_l \bar{u}_{N,l}(x, y)$$

alakban, ahol L az η_l kollokációs pontok száma a Γ_N^I – azaz a diszkrétizált, sarokpontokat nem tartalmazó – peremen. Az $\bar{u}_{N,l}$ függvényeket a

$$\begin{aligned} \frac{\partial^2 \bar{u}_{N,l}(x_i, y_j)}{\partial x^2} + \frac{\partial^2 \bar{u}_{N,l}(x_i, y_j)}{\partial y^2} - \sigma \bar{u}_{N,l}(x_i, y_j) &= f(x_i, y_j), \quad (x_i, y_j) \in \Omega_N \\ \bar{u}_{N,l}|_{\eta_m} &= \delta_{lm}, \quad \eta_m \in \Gamma_N^I. \end{aligned} \quad (5.19)$$

peremérték-feladatok megoldásával kapjuk meg, ami szintén elvégezhető diagonalizálással. A ξ_l ($l = 1, \dots, L$) konstansokat úgy számítjuk ki, hogy az (5.13) teljesüljön Γ_N^I -n, azaz

$$\begin{aligned} \alpha|_{\eta_m} \left(\tilde{u}_N|_{\eta_m} + \sum_{l=1}^L \xi_l \bar{u}_{N,l}|_{\eta_m} \right) + \beta|_{\eta_m} \left(\frac{\partial \tilde{u}_N}{\partial n} \Big|_{\eta_m} + \sum_{l=1}^L \xi_l \frac{\partial \bar{u}_{N,l}}{\partial n} \Big|_{\eta_m} \right) &= f|_{\eta_m}, \\ m &= 1, \dots, L \end{aligned} \quad (5.20)$$

amely felírható az

$$\mathbf{M}\Xi = \tilde{\mathbf{E}}$$

egyenletrendszerként, ahol $\mathbf{M}$ a hatásmátrix. Itt

$$\begin{aligned} \Xi &= [\xi_1 \ \dots \ \xi_L]^T, \\ \mathbf{M} &= [m_{ij}]_{i,j=1}^L, \quad m_{ij} = \alpha|_{\eta_i} \bar{u}_{N,j}|_{\eta_i} + \beta|_{\eta_i} \frac{\partial \bar{u}_{N,j}}{\partial n} \Big|_{\eta_i}, \\ \tilde{\mathbf{E}} &= [e_i]_{i=1}^L, \quad \tilde{e}_i = f|_{\eta_i} - \alpha|_{\eta_i} \tilde{u}_N|_{\eta_i} - \beta|_{\eta_i} \frac{\partial \tilde{u}_N}{\partial n} \Big|_{\eta_i} \end{aligned} \quad (5.21)$$

A fő számítási igény az $L + 1$ Dirichlet-feladat megoldásából és az $\mathbf{M}$ mátrix invertálásából tevődik össze. Ez a módszer lassabb más módszereknél ha csak egyszer kell az (5.12)-(5.13) feladatot megoldani. Viszont az energiaegyenlet megoldásakor látni fogjuk, hogy azt minden

időpillanatban meg kell oldani, és mivel $\bar{u}_{N,l}$ ($l = 1, \dots, L$) nem függ az időtől, így azt egy előfeldolgozási lépésben elegendő kiszámítani. Ebben a lépésben szintén elvégezhető az M invertálása és tárolása.

A módszerrel a négyzet sarkaiban nem kapjuk meg a megoldást. Tekintsük a felső és az alsó oldal bal és jobb oldali végpontjait. Felírható a következő négy egyenlet, amelyből meghatározható a sarokponti négy ismeretlen:

$$\begin{aligned}
\alpha_t(x_0)u_{N,0} + \beta_t(x_0) \sum_{k=0}^N u_{N,k} D_{0k} &= g_t(x_0), \\
\alpha_t(x_N)u_{N,N} + \beta_t(x_N) \sum_{k=0}^N u_{N,k} D_{Nk} &= g_t(x_N), \\
\alpha_b(x_0)u_{0,0} + \beta_b(x_0) \sum_{k=0}^N u_{0,k} D_{0k} &= g_b(x_0), \\
\alpha_b(x_N)u_{0,N} + \beta_b(x_N) \sum_{k=0}^N u_{0,k} D_{Nk} &= g_b(x_N).
\end{aligned} \tag{5.22}$$

5.3. Időbeli diszkretizáció

Időfüggő parciális differenciálegyenletek diszkretizációja kétféleképpen történhet. Diszkretizálhatunk először időben és utána térben (Rothe-módszer), vagy először térben és utána időben (egyenesek módszere, *method of lines*) [1]. Rothe módszerénél minden időpillanatban egy időfüggetlen parciális differenciálegyenletet kell megoldani, míg a másik módszernél egy nagy méretű – sok esetben merev – közönséges differenciálegyenlet-rendszer kezdetiérték-feladatát. Időben magasabb rendű közelítésekre sok esetben külön programot kell írni a Rothe-módszernél, míg az egyenesek módszerénél felhasználhatjuk a rendelkezésre álló megoldókat, ráadásul az időbeli hiba is könnyen becsülhető és változó időlépés alkalmazható. Hátránya viszont, hogy mivel először térben diszkretizálunk, a hálót nem módosíthatjuk később. Rothe módszerénél minden időlépésben változtatható a háló, így adaptív módszert készíthetünk.

A dolgozatban Rothe módszerét követjük, azaz először időben, majd térben diszkretizálunk.

Alkalmazhatnánk spektrális közelítést az időre, mint változóra, de az túl nagy számítás-igénnyel jár. Létezik még exponenciális időintegrálás is, aminek előnye, hogy viszonylag kevés lépés elegendő a megfelelő pontosság eléréséhez [6]. A legtöbb esetben differencia módszereket használnak. Az időbeli diszkretizáció néhány csoportosítása:

I. Időbeli szintek szerint

- egylépéses
- töblépéses

II. Explicitég szerint

- explicit
- szemi-implicit (explicit-implicit)

- implicit

III. Közelítés rendje szerint

- elsőrendű
- másodrendű
- magasabb rendű

Az explicit módszereket főleg összenyomhatatlan folyadékoknál alkalmazzák, vagy pedig nagy Reynolds-számú összenyomhatatlan közegnél spektrálem közelítésekor [18]. A szemi-implicit eljárásnál a lineáris tagokat implicit, a nemlineáris tagokat explicit módon közelítik, így elkerülhető a nemlineáris egyenletrendszerhez szükséges iteratív megoldás. A teljesen implicit módszerek mind a lineáris, mind a nemlineáris tagokat implicit módon közelítik, így szükséges minden időlépésben még egy iteráció a nemlineáris egyenletrendszerhez. Merev differenciál-egyenleteknél ajánlatos ezt használni. További előnye, hogy egyes esetekben feltétel nélkül stabilak. A 4. táblázat a folyadékmechanikában leggyakrabban használt időbeli véges differencia diszkretizálásokat mutatja. A kétlépéses módszerek másodrendű, a háromlépésesek harmadrendű, a néglépésesek pedig negyedrendű közelítést biztosítanak. Az egylépéses módszereknél a szám jelzi, hogy hányadrendű a módszer. Az Euler-módszerek elsőrendűek.

	egylépéses	kétlépéses	háromlépéses	néglépéses
explicit	előretartó Euler RK2, RK3, RK4 RK3/CK, RK4/CK	AB2 AB/BDE2	AB3 AB/BDE3	AB4 AB/BDE4
szemi-imp.	szemi-imp. Euler RK2/CN, RK3/CN	AB/BDI2 AB/CN, LF/CN WAB/BDI2	AB/BDI3	AB/BDI4
implicit	retrográd Euler	BDI2	BDI3	BDI4

4. táblázat. Időbeli differenciálások gyakori típusai

A táblázatban alkalmazott jelölések:

- RK k : k -adrendű Runge-Kutta
- RK k /CK: Carpenter-Kennedy-féle k -adrendű Runge-Kutta
- RK k /CN: k -adrendű Runge-Kutta/Crank-Nicolson
- AB k : k -adrendű Adams-Bashforth
- AB/BDE k : k -adrendű explicit Adams-Bashforth/retrográd deriválás
- AB/BDI k : k -adrendű szemi-implicit Adams-Bashforth/retrográd deriválás
- AB/CN: Adams-Bashforth/Crank-Nicolson
- LF/CN: középpont-szabályon alapuló módszer/Crank-Nicolson
- WAB/BDI2: szemi-implicit súlyozott Adams-Bashforth/retrográd deriválás
- BDI k : k -adrendű retrográd deriválás

Összefoglalásul elmondható, hogy ha explicit módszert választunk, akkor az RK4/CK ajánlott, összenyomhatatlan folyadékokra pedig az AB/BDI k módszerek.

A dolgozatban a szemi-implicit Euler és az AB/BDI2 módszert használjuk. Az előbbi a

$$\frac{\partial u}{\partial t} = L(u) + N(u)$$

differentiálegyenletet (L lineáris, N nemlineáris differenciáloperátor)

$$\frac{u^{n+1} - u^n}{\Delta t} = L(u^{n+1}) + N(u^n) \quad (5.23)$$

módon, az utóbbi

$$\frac{3u^{n+1} - 4u^n + u^{n-1}}{2\Delta t} = L(u^{n+1}) + 2N(u^n) - N(u^{n-1}) \quad (5.24)$$

módon közelíti. A Δt az időlépés, a felső index azt jelzi, hogy melyik időszinten vesszük a függvényt: $u^{n+1}(\mathbf{x}) = u(\mathbf{x}, (n+1)\Delta t)$.

Az időlépés lehet akár változó is. Ennek előnye, hogy elég ott kisebb időlépéssel dolgozni, ahol a stabilitás megköveteli. A dolgozatban állandó időlépéssel számoltunk.

A többlépéses módszerek – szemben az egylépéses módszerekkel – nem önindítóak, ezért szükséges az első néhány lépést másképpen előállítani. Ennek egyik módja, hogy a szükséges időlépésekben egylépéses módszert használunk. Bizonyítható, hogy az indító módszer az általános sémánál eggyel alacsonyabb rendű is lehet, az időbeli pontosság megmarad [18]. Éppen ezért az ABBDI2-höz az első időlépést a szemi-implicit Euler-módszerrel állítjuk elő.

A (3.17) kapcsolt egyenletrendszerben először a hőmérsékletet határozzuk meg az energia-egyenletből, majd annak ismeretében oldjuk meg az ω -ra felírt mozgásegyenletet, végül pedig a harmadik egyenletből meghatározzuk ψ -t. A ψ ismeretében meghatározhatók az u és v alapváltozók (3.8)-ból. Tehát a szemi-diszkrét alak a szemi-implicit Euler-módszerrel:

$$\begin{aligned} \frac{T^{n+1} - T^n}{\Delta t} + \mathbf{v}^n \cdot \nabla T^n &= \frac{1}{\text{RePr}} \Delta T^{n+1}, \\ \frac{\omega^{n+1} - \omega^n}{\Delta t} + \mathbf{v}^n \cdot \nabla \omega^n &= \frac{1}{\text{Re}} \Delta \omega^{n+1} + \text{Ri} \frac{\partial T^n}{\partial x}, \\ \Delta \psi^{n+1} + \omega^{n+1} &= 0. \end{aligned}$$

Átalakítások után:

$$\begin{aligned} \Delta T^{n+1} - \sigma_T T^{n+1} &= f_T^n, \\ \Delta \omega^{n+1} - \sigma_\omega \omega^{n+1} &= f_\omega^n, \\ \Delta \psi^{n+1} + \omega^{n+1} &= 0, \end{aligned} \quad (5.25)$$

ahol

$$\begin{aligned} \sigma_T &= \frac{\text{RePr}}{\Delta t}, \quad f_T^n = \text{RePr} \left(\mathbf{v}^n \cdot \nabla T^n - \frac{T^n}{\Delta t} \right), \\ \sigma_\omega &= \frac{\text{Re}}{\Delta t}, \quad f_\omega^n = \text{Re} \left(\mathbf{v}^n \cdot \nabla \omega^n - \frac{\omega^n}{\Delta t} - \text{Ri} \frac{\partial T^n}{\partial x} \right). \end{aligned}$$

Az (5.24) alapján az AB/BDI2 módszerrel diszkretizált egyenletek alakja ugyanaz, mint az (5.25) egyenletek, de az együtthatók és a differenciálegyenletek jobb oldalai különböznek:

$$\sigma_T = \frac{3\text{RePr}}{2\Delta t}, \quad f_T^n = \text{RePr} \left(2\mathbf{v}^n \cdot \nabla T^n - \mathbf{v}^{n-1} \cdot \nabla T^{n-1} + \frac{-4T^n + T^{n-1}}{2\Delta t} \right),$$

$$\sigma_\omega = \frac{3\text{Re}}{2\Delta t}, \quad f_\omega^n = \text{Re} \left(2\mathbf{v}^n \cdot \nabla \omega^n - \mathbf{v}^{n-1} \cdot \nabla \omega^{n-1} + \frac{-4\omega^n + \omega^{n-1}}{2\Delta t} - \text{Ri} \frac{\partial}{\partial x} (2T^n - T^{n-1}) \right)$$

Az (5.25) alapján minden időpillanatban meg kell oldani két Helmholtz-egyenletet és egy Poisson-egyenletet.

5.4. Térbeli diszkretizáció

A nehézség az (5.25) megoldásában az, hogy ω -ra nincs megadva peremfeltétel. Ezt a problémát a már korábban is alkalmazott hatásmátrix technikával kerüljük meg. Miután egy időlépésben kiszámítottuk a hőmérsékletet, a megoldandó két differenciálegyenlet a hozzájuk tartozó peremfeltételekkel:

$$\begin{aligned} \Delta\omega - \sigma\omega &= F, \quad \mathbf{x} \in \Omega, \\ \Delta\psi + \omega &= 0, \quad \mathbf{x} \in \Omega, \\ \psi &= g, \quad \frac{\partial\psi}{\partial n} = h, \quad \mathbf{x} \in \Gamma, \end{aligned} \tag{5.26}$$

ahol bevezettük az egyszerűsítő $\omega := \omega^{n+1}$, $\psi := \psi^{n+1}$, $F := f_\omega^n$, $\sigma := \sigma_\omega$ jelöléseket, továbbá g és h a (3.14) és (3.15)-nek megfelelő, $t = (n+1)\Delta t$ -beli peremfeltételek.

A megoldást az (5.4)-beli Chebyshev kollokációs pontokban keressük. Az (ω, ψ) megoldást (ω_N, ψ_N) -nel közelítjük, ahol $\omega_N \in \mathbb{P}_N$, $\psi_N \in \mathbb{P}_N$. A hatásmátrix módszert hasonlóan alkalmazzuk, mint az 5.2. fejezetben. A [18] könyvben részletesen megtalálható származtatás végeredménye a következő:

a differenciálegyenletek lineárisak, ezért írható $\omega_N = \tilde{\omega}_N + \bar{\omega}_N$ és $\psi_N = \tilde{\psi}_N + \bar{\psi}_N$. Az (5.26) peremérték-feladathoz a $\tilde{P}$ -probléma:

$$\begin{aligned} \Delta\tilde{\omega}_N - \sigma\tilde{\omega}_N &= F, \quad (x_i, y_j) \in \Omega_N \\ \tilde{\omega}_N &= 0, \quad (x_i, y_j) \in \Gamma_N^I \end{aligned} \tag{5.27}$$

és

$$\begin{aligned} \Delta\tilde{\psi}_N &= -\tilde{\omega}_N, \quad (x_i, y_j) \in \Omega_N \\ \tilde{\psi}_N &= g, \quad (x_i, y_j) \in \Gamma_N^I. \end{aligned} \tag{5.28}$$

A $\bar{P}$ -feladat megoldásához az $\bar{\omega}$ és $\bar{\psi}$ függvényeket alapmegoldások lineáris kombinációjaként írjuk fel:

$$\bar{\omega}_N = \sum_{l=1}^L \xi_l \bar{\omega}_l, \quad \bar{\psi}_N = \sum_{l=1}^L \xi_l \bar{\psi}_l. \tag{5.29}$$

Ezzel a $\bar{\mathcal{P}}_l$ -feladat:

$$\begin{aligned}\Delta\bar{\omega}_l - \sigma\bar{\omega}_l &= 0, & (x_i, y_j) &\in \Omega_N \\ \bar{\omega}_l|_{\eta_m} &= \delta_{lm}, & \eta_m &\in \Gamma_N^I\end{aligned}\tag{5.30}$$

és

$$\begin{aligned}\Delta\bar{\psi}_l &= -\bar{\omega}_l, & (x_i, y_j) &\in \Omega_N \\ \bar{\psi}_l|_{\eta_m} &= 0, & \eta_m &\in \Gamma_N^I.\end{aligned}\tag{5.31}$$

Hátra van még a $\xi_1, \dots, \xi_L$ meghatározása. Ezeket úgy választjuk meg, hogy a Neumann-peremfeltétel teljesüljön, azaz

$$\sum_{l=1}^L \frac{\partial \bar{\psi}_l}{\partial n} \Big|_{\eta_m} \xi_l = \left(h - \frac{\partial \tilde{\psi}_N}{\partial n} \right) \Big|_{\eta_m}, \quad \eta_m \in \hat{\Gamma}_N^I \subset \Gamma_N^I, \quad m = 1, \dots, L,\tag{5.32}$$

amely mátrixos alakban az

$$\mathbf{M}\Xi = \tilde{\mathbf{E}}\tag{5.33}$$

módon írható, ahol $\mathbf{M}$ a hatásmátrix. Az 5.2. részben láttuk, hogy L a perempontok száma Γ_N^I -n. Ekkor azonban $\mathbf{M}$ -nek négy nulla sajátértéke van, így nem invertálható [18]. Négy kollokációs pont eltávolításával $\mathbf{M}$ reguláris; ezek a pontok:

$$P_1(x_1, -1), \quad P_2(x_{N-1}, -1), \quad P_3(x_1, 1), \quad P_4(x_{N-1}, 1).\tag{5.34}$$

Így (5.32)-ben a $\hat{\Gamma}_N^I$ halmaz a Γ_N^I és az (5.34) különbsége.

Az örvényesség nincs egyértelműen meghatározva a peremen. Ahhoz, hogy ω_N -et megkapjuk Γ_N^I -n, felhasználjuk az örvényesség (3.9)-beli definícióját.

Az (5.26) megoldásának menete az alábbi:

I. Előfeldolgozás

1. Az $\bar{\omega}_l, \bar{\psi}_l$ alapmegoldások kiszámítása az (5.30), illetve (5.31) egyenletekből.
2. Az $\mathbf{M}$ felépítése az (5.32)-ből, majd az inverzének tárolása

II. Minden időlépésben

1. Az (5.26) első differenciálegyenletének jobb oldalát (F) meghatározzuk. Ez az előző időlépésekben kiszámított függvényértékeket tartalmazza
2. Az $\tilde{\omega}_N$ és $\tilde{\psi}_N$ meghatározása (5.27)-ből, illetve az (5.28)-ból.
3. Az (5.33)-ben $\tilde{\mathbf{E}}$ kiszámítása
4. A $\Xi = [\xi_1 \dots \xi_L]^T$ meghatározása az (5.33)-ból.
5. A $\xi_l, \bar{\omega}_l$ és $\bar{\psi}_l$ ismeretében (5.29) alapján $\bar{\omega}$ és $\bar{\psi}$ kiszámítása
6. Az ω_N és ψ_N meghatározása a $\tilde{\mathcal{P}}$ és $\bar{\mathcal{P}}$ feladatok eredményeinek összeadásával.
7. ω_N meghatározása a diszkretizált peremen: $\omega_N = -\Delta\psi_N$.

5.5. Leállási feltétel, konvergencia

Ha állandósult állapotot akarunk meghatározni, akkor szükséges egy leállási feltétel, ami szabályozza, hogy mikor fejezzük be az időbeli integrálást. A relatív L_2 hibát vettük alapul, azaz egy $u(\mathbf{x})$ függvényre:

$$E_N(u) = \frac{\|u - u_N\|_2}{\|u\|_2} \approx \frac{\|u_N^{n+1} - u_N^n\|_2}{\|u_N^{n+1}\|_2}, \quad (5.35)$$

ahol u a pontos, u_N a közelítő megoldás.

Létezik néhány szabály, ami segít annak eldöntésében, hogy a kapott eredmény megfelelő közelítést ad-e. Közöséges differenciálegyenletekre [3]:

- A csonkítási hiba nagyságrendje megegyezik az utolsó megtartott spektrális együttható nagyságrendjével:

$$E_T(N) \approx \mathcal{O}(|a_N|).$$

- Ha az a_N -hez közel lévő, de a_N -nél kisebb együtthatók abszolút értékben monoton csökkenő módon tartanak a_N -hez, akkor az eredmény numerikus tapasztalatok alapján valószínűleg helyes.
- További ellenőrzési módszer, hogy a számítást különböző N felosztásra végezzük el.

A (4.8) közelítés miatt az a_{mn} együtthatókat az indexek összegének függvényében kell vizsgálni a fentiek szerint.

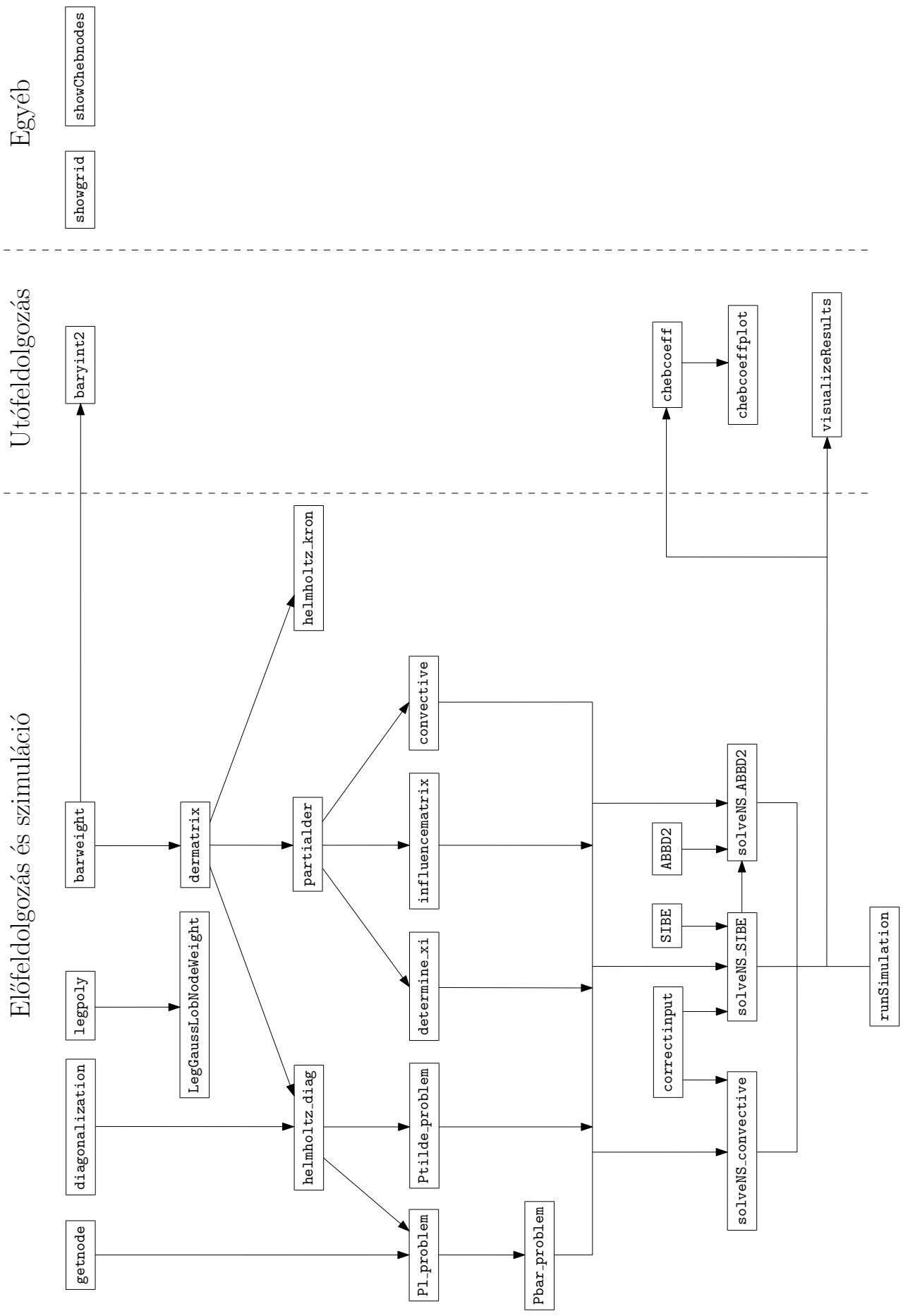
6. A MATLAB programok felépítése

Minden programot a MATLAB 2011a verziójában készítettünk el. A fejezetben vázoljuk a programok kapcsolatát, megmutatjuk, hogy milyen célt szolgálnak, végül pedig azt, hogy milyen elveket vettünk figyelembe az elkészítésükkor.

A beépített függvények neveit *dőlt*, míg a saját programokat *írógép* stílussal szedtük. Ebben a fejezetben függvényen a MATLAB-ben *function*-ként definiált szerkezetet értjük.

Az 5.1. fejezetben láttuk, hogy egy $N \times N$ -es mátrix akkor és csak akkor diagonalizálható, ha N darab lineárisan független sajátvektora van. Ez ellenőrizhető rangszámvizsgálattal, amit a MATLAB szinguláris érték felbontással végez (l. *rank*). Kiszámítottam a rangot $N = 2, \dots, 100$ mátrix méretig és azt találtam, hogy $D^{(2)}$ diagonalizálható.

A 6. ábra a MATLAB programok kapcsolatát jelzi. Egy adott függvényhez befutó nyilak azt mutatják, hogy milyen korábban definiált függvényeket használ fel (csak a saját függvényeket mutatjuk, a beépített MATLAB függvényeket nem). Az ábrán jelzett programok a `runSimulation` és a `TODO` szkripteket leszámítva függvények. Biztonsági okokból nem használtunk *global* és *persistent* változókat. A programok jól dokumentáltak és a MATLAB által gyakran alkalmazott fejléceket használtuk. A 6. ábra első oszlopában látszik, hogy három programból nem indul ki nyíl. Az egyik a legmagasabb szinten lévő `runSimulation` szkript, amellyel a felhasználó közvetlenül kapcsolatban van. A másik, a `helmholtz_kron` nem játszik szerepet a programfutásban, viszont jól mutatja (l. 3. táblázat), hogy mennyivel lassabb a Kronecker-szorzáttal való közelítés a diagonalizálásnál (l. `helmholtz_diag`). A `legpoly` és a `LegGaussLobNodeWeight` függvényeket szintén nem használjuk fel a kollokációs közelítéshez, de a későbbiekben megvalósítandó Legendre spektrálem módszer alapját képezik. A 6. ábra középső oszlopában az utófeldolgozással kapcsolatos függvények állnak. Az utolsó oszlopban a szemléltetést segítő `showgrid` és `showChebnodes` függvények és a további tennivalók listája látható.



6. ábra. A MATLAB programok kapcsolata

Röviden bemutatjuk a készített programokat. A függelékben csatolt programok tartalmazzák a hivatkozást, amelyek segítségével az algoritmusok készültek.

barweight

Teljesen vektorizált módon kiszámítja az adott ponthalmazhoz tartozó baricentrikus súlyokat.

diagonalization

A Helmholtz-operátor teljes diagonalizálását végzi el descartes-i koordináta-rendszerben (l. 5.1. fejezet). A függvény alkalmas más koordináta-rendszerekre is; akkor $D^{(2)}$ módosul.

getnode

Mind a Helmholtz-egyenlet Robin-féle peremfeltételeinél (l. (5.19)), mind a Navier-Stokes egyenletnél (l. (5.30), (5.31)) láttuk, hogy a $\bar{P}_l$ -feladatnál a peremgörbe (most a négyzet oldalai) pontjait kellett kiválasztani óramutató járásával megegyező irányban. Ez a függvény megkeresi, hogy az aktuális pont melyik oldalon van; ott 1 értéket ad, a többi ponthoz pedig a 0-t rendeli.

SIBE

A Navier-Stokes egyenlet szemi-implicit Euler-módszerrel diszkretizált változatának jobb oldalát tartalmazza.

ABBD2

A Navier-Stokes egyenlet másodrendű szemi-implicit Adams-Bashforth/retrográd deriválás módszerrel diszkretizált változatának jobb oldalát tartalmazza.

dermatrix

A baricentrikus súlyok ismeretében meghatározza az elsőrendű deriváló mátrixot, majd rekurzíóval a magasabb rendű deriváló mátrixokat. Létezik egzakt formula az első- és másodrendű Chebyshev deriváló mátrixok generálására, viszont az itt alkalmazott módszer bármilyen ponthalmazra használható. A függvényben egy stabil eljárás szerepel, mégis a magasabb rendű deriváltak közelítések a kerekítési hibák hatványozottan nőnek. A másodrendű deriváló mátrixnál fellépő kerekítési hiba $N < 100$ Legendre-vagy Chebyshev-pont esetén elhanyagolható. Ilyen nagy foksámú közelítés pedig szinte minden esetben elegendő.

helmholtz_kron

Az (5.1)-(5.2) típusú peremérték-feladatot oldja meg Kronecker-féle tenzorszorzattal.

partialder

Egy kétváltozós függvény elsőrendű parciális deriváltjait számítja ki. Mivel ez a függvény minden időpillanatban meghívásra kerül, ezért a rácsot és a deriváló mátrixot bemenetként kell megadni. Minimális módosítással létrehozható önmagában használható változat, akkor a rácsot és a deriváló mátrixot a függvényen belül kell létrehozni.

helmholtz_diag

Az (5.1)-(5.2) típusú peremérték-feladatot oldja meg diagonalizálással. Alapvető megoldó, mivel a szemi-implicit időbeli diszkretizálás során Helmholtz-egyenleteket (illetve annak speciális eseteként Poisson-egyenletet) kapunk.

convective

Az $\mathbf{u} \cdot \nabla g$ szorzatot határozza meg, ahol g egy skalár függvény.

determine_xi

A ξ_l ($l = 1, \dots, 4N - 8$) értékeket az (5.33) lineáris egyenletrendszerből határozza meg az influencematrix-tól kapott M^{-1} segítségével. Az egyenletrendszer jobb oldalát az (5.32)-ből veszi.

influencematrix

A hatásmátrix inverzét adja meg kimenetként. A hatásmátrixot az (5.32) egyenletből határozzuk meg, majd csökkentjük a méretét négygyel, úgy, hogy az (5.34) pontoknak megfelelő négy oszlopot és sort töröljük. Az inverz képzése $\mathcal{O}((4N - 8)^3)$ régi flop műveletigénnyel jár, de elegendő egyszer elvégezni.

Ptilde_problem

Az (5.16) $\tilde{\mathcal{P}}$ -feladatot oldja meg.

Pl_problem

Az (5.30), (5.31) peremérték-feladatokat oldja meg. Az aktuális perempontot a `getnode` függvénnyel választjuk ki, majd a $8(N - 1)^2$ számú Helmholtz-egyenletet diagonalizálással oldjuk meg. Az időtől való függetlenség miatt elegendő egyszer, egy előfeldolgozási lépésben elvégezni ezt a nagy számításigényű részfeladatot.

Pbar_problem

Az (5.29)-beli $\bar{\mathcal{P}}$ -feladat a $\bar{\mathcal{P}}_l$ -feladatok összegzésével jár és az $\bar{\omega}_l$, $\bar{\psi}_l$ és ξ_l ($l = 1, \dots, N - 8$) bemenet ismeretében történik.

solveNS_SIBE

Az időben szemi-implicit Euler módszerrel diszkretizált egyenleteket oldja meg az 5.4. rész végén jelzett folyamatot követve. Bemenetként az ismert külső erőrendszer intenzitásvektorát, a peremfeltételeket, a kezdeti feltételeket és a kinematikai viszkozitást kéri. Opcionális bemenet az időlépés, a maximális iterációszám és a leállási feltétel. Kimenetként minden időpillanatban megadja az ω , ψ , u és v függvényeket, továbbá lekérdezhetjük a futáshoz szükséges iterációk számát, az így elért relatív hibát és a rácsot. Mivel a `solveNS_ABBD2` megoldó az első időlépést ezen megoldó segítségével végzi, ezért az `solveNS_ABBD2`-höz szükséges biztosítanunk ω^0 -t és $\mathbf{v}^0 \cdot \nabla \omega^0$ -t. Nagy stabilitása miatt ez az alapértelmezett megoldó.

solveNS_ABBD2

Az AB/BDI2 módszerrel diszkretizált feladatot oldja meg. Időben másodrendben pontos, de tapasztalataink alapján a szemi-implicit Euler-módszernél kevésbé stabil.

runSimulation

Interaktív módon lehet feladatokat megoldani vele. A felhasználóval a parancssoron keresztül kommunikál és a kívánt opciókkal hívja meg a függvényeket.

barint2

Baricentrikus Lagrange-interpolációt valósít meg két dimenzióban tenzorszorzat rácson a (4.10) alapján. A gyors futás érdekében ezt is vektorizáltuk a `bsxfun`-nal. Hasonlóan működik, mint az egy dimenziós változat [7].

²A sarokpontok nélkül $4N - 4$ darab perempont van és ω_l és ψ_l is meghatározandó.

`chebcoeff`

A (4.8) csonkított sorfejtés $a_{i,j}$ együtthatóit határozza meg gyors Chebyshev transzformációval. A Chebyshev-polinomok (4.1)-beli trigonometrikus alakja miatt a diszkrét Chebyshev-transzformáció elvégezhető gyors Fourier-transzformációval a mátrix-vektor szorzásnál kisebb műveletigénnyel. A `chebcoeff` egy utasítással bővíti a <http://www.mathworks.com/matlabcentral/fileexchange/4416-2d-chebyshev-transform> helyről letölthető kódot. Ugyanis a (4.2) nálunk negatív előjellel szerepel, ezért a bemenő mátrix oszlopait és sorait fel kell cserélnünk.

`chebcoeffplot`

Egy négyzetes mátrix elemeinek abszolút értékét ábrázolja a hozzá tartozó indexek függvényében fél-logaritmikus skálán. A 14, 16 és 18. ábrát ezen függvény segítségével kaptuk.

`legpoly`

Az első-, másod-, ... N -edfokú Legendre-polinomot és deriváltját kiértékeli az adott pontokban. Az eredmény az L és Lp mátrix, melyre

$$L_j(x_i) = L_{ij}, \quad \frac{dL_j(x_i)}{dx} = Lp_{ij},$$

ahol L_j a j -edfokú Legendre-polinom.

`LegGaussLobNodeWeight`

A Legendre-Gauss-Lobatto (LGL) csomópontokat és súlyokat határozza meg a megadott pontosság mellett. Az LGL pontok az $(1 - x^2)L'_N(x)$ függvény zérushelyei, és egzakt formula nem létezik a kiszámításukra. Közelítő meghatározásuk történhet egy sajátértékfeladat vagy pedig nemlineáris egyenletek megoldásával. Az előbbi módszer nagyobb N értékekre túl nagy kerekítési hibával jár, ezért az utóbbit implementáltuk. A Newton-iteráció kezdőértékének az

$$x_j^0 = -\cos\left(\frac{(j+0,25)\pi}{N} - \frac{3}{8\pi N} \frac{1}{j+0,25}\right), \quad j = 1, \dots, N-1$$

számokat választottuk [13], de más kezdővektor is megadható (l. [22]). A zérushelyek és súlyok ezután a hivatkozott két könyvben megtalálható módon számíthatók.

`visualizeResults`

Kétdimenziós adatok megjelenítését végzi. Lehet készíteni felületet (*surf*), koordinátavonalas felületet (*mesh*), kontúrvonalas ábrát (*contour*), áramvonal megjelenítést (*streamslice*), vektorábrát (*quiver*) és vektorábrát színezéssel (*quiverc*³). A bemenetek ellenőrzését az *inputParser* osztály végzi. Lehetőség van a *quiver* esetén a sebesség normalizálására is, így egyenlő nagyságú nyilat lehet kirajzolni. Mivel a *quiverc* nagyon erőforrás-igényes, ezért $N > 50$ felett már nem engedélyezzük a használatát.

`showgrid`

Az elsőfajú Chebyshev-polinomok szélsőértékhelyiből alkotott tenzorszorzat rácsot rajzolja ki. Ezzel készült a 2. ábra.

³Letölthető a <http://www.mathworks.com/matlabcentral/fileexchange/3225-quiverc> honlapról.

showChebnodes

Az említett szélsőérték helyeket geometriailag úgy kapjuk, hogy az origó középpontú egység-sugarú félkör ívét egyenlő részekre osztjuk, majd merőlegesen a vízszintes tengelyre vetítjük. Ezt szemlélteti a `showChebnodes` függvény.

correctinput

A numerikus tömbként vagy pedig *function handle*-ként megadott peremfeltételeket hozza olyan alakra, hogy azt a további függvények kezelni tudják.

TODO

Tennivalók listája.

6.1. Az algoritmusok megvalósításának szempontjai

A programkód megírása közben elsődleges szempont volt a modularitás. Ahogy egyre bonyolultabbá vált, egyre fontosabb lett, hogy a későbbiekben könnyen bővíthető legyen. Az említett függvények egy-egy jól behatárolható feladat elvégzésére szolgálnak. A Reynolds-szám növekedtével jelentősen nőtt a stabilitás szempontjából szükséges időlépések száma. Éppen ezért a futásidő tekintetében fontos, hogy ami időfüggetlen, azt egy előfeldolgozási lépésben számítsuk ki, ne pedig minden időpontban.

Az általunk alkalmazott spektrál módszer sem térben, sem időben nem adaptív. Az időbeli adaptivitás viszonylag könnyen megvalósítható (l. pl. [18]) és nem befolyásolja a rácsot. Azért döntöttünk a nem adaptív rács alkalmazása mellett, mert így elegendő egyszer létrehozni és kiszámítani a szükséges mennyiségeket, nem pedig minden időpillanatban. A Chebyshev-együtthatók utalnak a diszkretizációs hibára.

A programok futtatásához elegendő (akár egy sok évvel korábbi) MATLAB, nem szükséges hozzá semelyik Toolbox. A spektrál módszerekre jellemzően a megoldandó egyenletrendszerek kis méretűek, így nem érdemes GPU-t használni, mert a VRAM-ból a RAM-ba történő adatátvitellel együtt már hosszabb a futásidő, mintha a CPU-n oldanánk meg. Az időlépéses iterációban a pillanatnyi függvényértékek a korábbi ciklusbeli értéktől függenek, ezért a programnak ez a része nem, vagy csak nehezen párhuzamosítható. Ennek ellenére a megoldóink a rendelkezésre álló összes processzormagot és szálát kihasználják az implicit többszálúság és a vektorizált kód miatt. A MATLAB-ben sok lineáris algebrai függvény alacsony szintű, gyors C-kódként van optimalizálva, amelyek ráadásul kihasználják a BLAS és LAPACK rutinokat, ezáltal külön utasítás nélkül is több szálon történő futást tesznek lehetővé – ez az implicit többszálúság. Az ezt támogató, általunk használt beépített függvények az *eig*, *diag*, *inv*, *bsxfun*, *sum*, *isnan*, stb. és a lineáris egyenletrendszer direkt megoldásához használt `\` operátor. Az újabb MATLAB verziók rendelkeznek JIT fordítóval, de sok esetben még így is megéri vektorizálni. A vektorizálás során a *for* és *while* ciklusokat kicseréljük a velük ekvivalens vektor és mátrix műveletekre. A MATLAB Profiler segítségével megmértük, hogy mely függvények teszik ki a futásidő jelentős részét; ezek optimalizálására külön figyelmet fordítottunk. Nem vektorizált például a `legpoly` program, ugyanis a futásidőre nézve elhanyagolható és *for* ciklussal áttekinthetőbb. Amikor sok mátrixot kellett tárolni (például az $\omega(x_i, y_j, t_k)$ függvényértékeket),

akkor ahhoz az áttekinthetőség miatt nem hipermátrixot, hanem cella tömböt használtunk. A MATLAB programok gyorsításának egy további lehetősége, ha MEX-függvényeket írunk. A MEX lényege, hogy a hosszan futó MATLAB programot átírjuk C-re, majd MEX-függvényként hívjuk meg a MATLAB-ből. Az implicit többszálúsítás miatt ennek jelen esetben nem volt jelentősége.

Az **A** függelékben látható módon a programok nagy része mátrixszorzásra és mátrixok Hadamard-szorzatára (MATLAB-ben: `.*`) van visszavezetve.

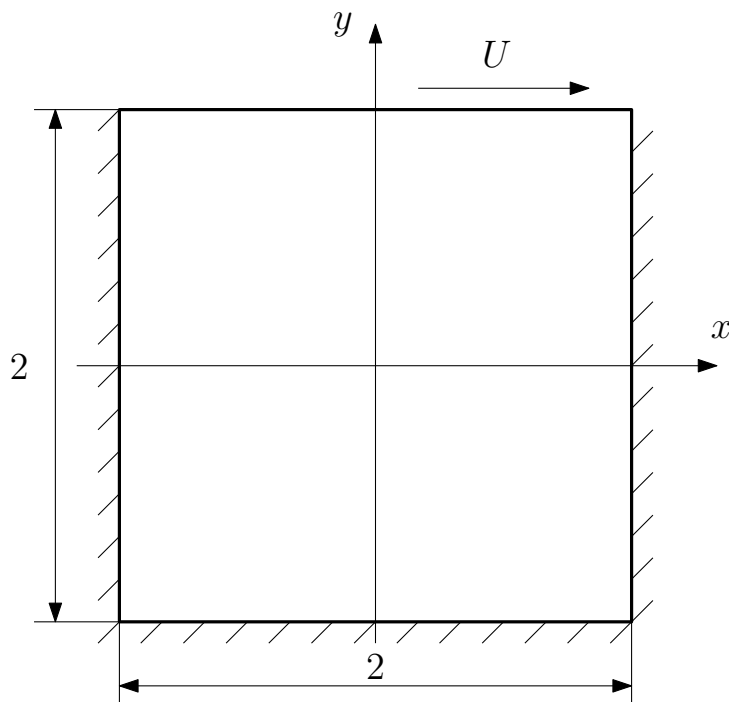
7. Numerikus példák

Az elkészített programokat különböző feladatokon teszteljük. Vegyük először az egyik leghíresebb tesztfeladatot, a „lid-driven cavity”-t, amelyen részletesen bemutatjuk a megoldás menetét.

7.1. „Lid-driven cavity”

A feladattal modellezhető egy téglatest alakú tartályban kialakuló áramlás. Vegyük a 7. ábrán látható üreget, melyet teljesen kitölti a folyadék. A folyadék a felső lap állandó U sebességű vízszintes mozgásától áramlásba jön. A falak mentén nincs csúszás, a gravitációt és más külső erőrendszert nem vesszük figyelembe.

A négyzet felső két sarka szingularitási pont, mert ott a vízszintes irányú sebességkoordináta nem egyértékű. Látni fogjuk, hogy ez jelentősen rontja a konvergenciasebességet és az időbeli diszkretizációs sémák stabilitást is. A kialakuló áramlást a Reynolds-szám nagysága befolyásolja.



7. ábra. Az üreg geometriája

A feladat jellegéből következően a peremfeltételek:

$$u(x, 1) = U, \quad u(-1, y) = u(x, -1) = u(1, y) = v(-1, y) = v(x, -1) = v(1, y) = v(x, 1) = 0.$$

A (3.14) egyenlet alapján $\psi = 0$ mind a négy peremen, mert nincs normális irányú áramlás. A (3.15) szerint a felső peremen $\partial\psi/\partial y = U$. Fentebb hallgatólagosan azt feltételeztük, hogy a peremfeltételek minden időpillanatban ugyanazok. A kompatibilitási feltétel (l. [18]) értelmében a kezdeti sebességmezőnek is ki kell elégítenie a peremfeltételeket. Tehát nem lehet

nyugalomban a folyadék a fenti peremfeltételek mellett. Bár fizikailag értelmes lenne a

$$\mathbf{v}_0(\mathbf{x}, t) = \begin{cases} U \mathbf{i}, & \{-1 \leq x \leq 1, y = 1\} \\ 0, & \text{egyébként} \end{cases}$$

kezdősebesség, de ez nem deriválható és így ω_0 sem képezhető. Így a kompatibilitási egyenletet nem teljesítő $\mathbf{v}_0 \equiv 0$ kezdősebességet vesszük, amelyről látni fogjuk, hogy helyes eredményt ad.

Ha $\nu \rightarrow \infty$, azaz $Re \rightarrow 0$, akkor a Stokes-egyenletet kapjuk, tehát a Navier-Stokes megoldóval ezt is lehet kezelni. A `runSimulation` futtatásával interaktív módon tudjuk megoldani a feladatot, itt a szemi-implicit Euler-módszerrel való közvetlen megoldást mutatjuk meg. Vegyünk $\nu = 0,01 \text{ m}^2/\text{s}$ kinematikai viszkozitást, legyen $U = 1 \text{ m/s}$ (így $Re = 200$), az (5.35)-nek megfelelő leállási feltétel 10^{-9} , a maximális iterációs szám 2000, továbbá az időlépés $\Delta t = 0,1$. Legyen $\mathbf{v}_0 = 0 \text{ m/s}$ és $N = 20$ -adfokú Chebyshev-polinommal közelítünk mindkét koordináta irányban.

Írjuk be a parancssorba:

```
>> nu = 0.01;
>> N=20;
>> u0 = zeros(N+1); v0 = u0;
>> maxIteration = 2000;
>> reltol = 1e-9;
>> Dt = 0.1;
>> f = @(x,y) 0;
>> topBC = @(x) 1;
>> otherBC = @(x) 0;
>> [omega psi u v stat X Y] = solveNS_SIBE(N,f,otherBC, ...
      otherBC,otherBC,otherBC,otherBC,topBC,otherBC, ...
      otherBC,u0,v0,Dt,nu,maxIteration,reltol);
```

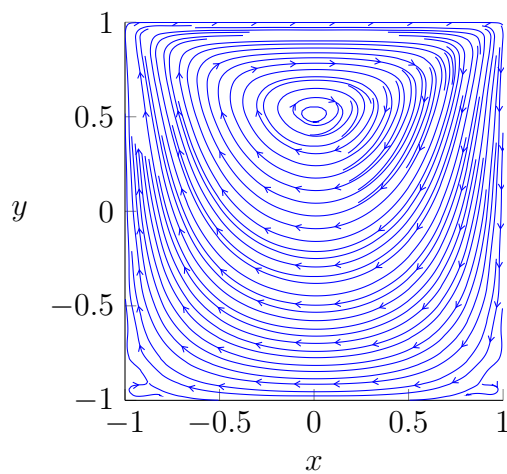
A program 978 lépés alatt végzett, tehát az állandósult állapot közel 97,8 másodperc alatt állt be. Az alkalmazott szemi-implicit Euler-módszer időben csak első rendben pontos, viszont minket az állandósult állapot érdekel, nem pedig az, hogy pontosan mikor következik be. A megoldást az ω , ψ , u és v függvényekre cella tömbökben kaptuk meg, például az $u(1, -1, 100\Delta t)$ értéke az `u{100*Dt}(N+1,1)` utasítással hívható elő.

A `visualizeResults` függvényvel sokféle ábra készíthető, most az áramvonalakat jelenítjük meg a

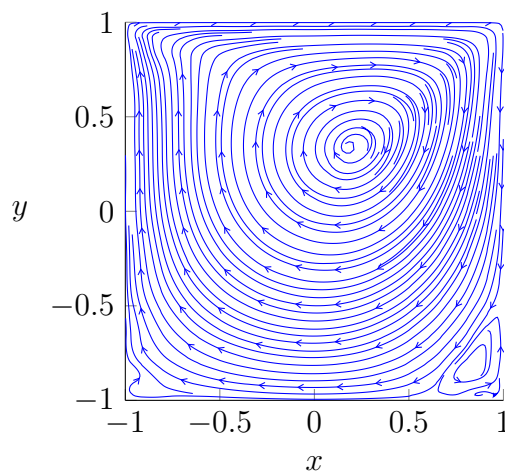
```
>> visualizeResults(X,Y,u{iter},v{iter},'method', ...
      'streamslice');
```

utasítással. Ekkor a 9. ábrán látható képet nyerjük. A 8. ábra a Stokes-egyenlet megoldását mutatja. Végtelen nagy viszkozitást feltételezve $u(-x, y) = u(x, y)$ és $v(-x, y) = -v(x, y)$.

Ha $Re > 0$, akkor u és v nem rendelkeznek a fentebbi paritási tulajdonságokkal. A Navier-Stokes egyenlet megoldásakor látható, hogy a jobb alsó sarokban kialakul egy másodlagos, a bal alsó sarokban pedig egy harmadlagos örvény.



8. ábra. Áramvonalak, $Re = 0$



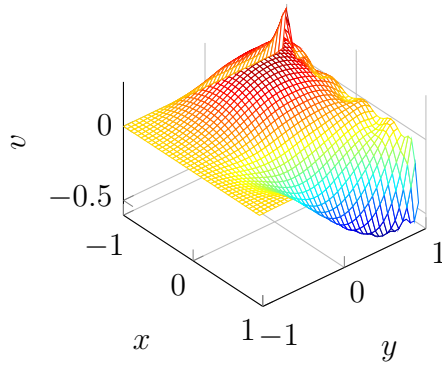
9. ábra. Áramvonalak, $Re = 200$

Az 5.5. részben láttuk, hogy a Chebyshev-együtthatókból lehet következtetni a megoldás helyességére. A Stokes-egyenletnél a szemlélet alapján is arra gondolhatunk, hogy u szimmetrikus az x tengelyre. Tehát $a_{i,j}$ -nek (i páros, $j = 0, \dots, N$) nullának kell lennie. A sejtést megerősíti a `>> coeffs = chebcoeff(A)` utasítás eredménye is: a spektrális együtthatók mátrixának minden második oszlopa gépi nulla. Ugyanúgy páros x -ben az ω és a ψ is.

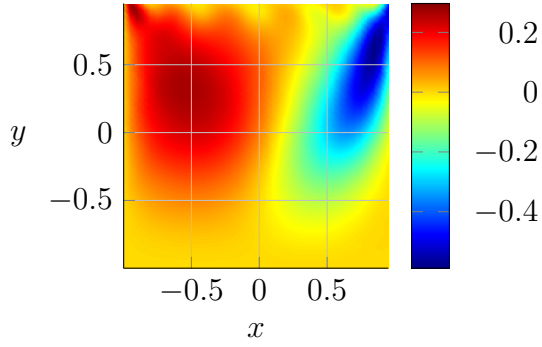
Hasznos lehet (ábrázoláshoz, irodalomban szereplő egyenközű rácsokon kapott értékekkel való összehasonlítás érdekében) a kapott függvényértékek interpolálása. A fenti példán ez a következő parancsokkal történik:

```
>> x = -cos(pi*(0:N)/N)';
>> newGrid = -0.999:0.05:0.999;
>> [new_v Xnew Ynew] = baryint2(x,v{iter},newGrid);
>> visualizeResults(Xnew,Ynew,new_v,'method','surf');
```

A v -re eredményül a 11. ábrán látható eloszlást kapjuk (felülről ránézve). A hálószerű felületet a 10. ábra szemlélteti.

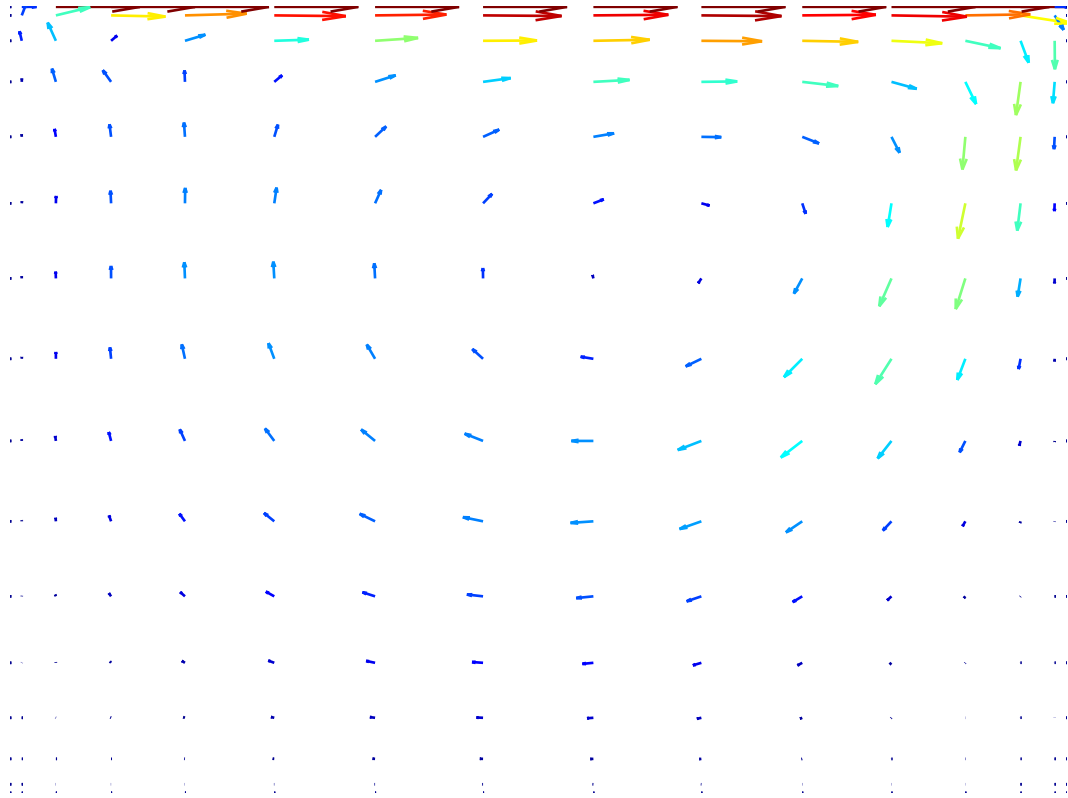


10. ábra. Az interpolált v sebességkoordináta képe, $Re = 200$



11. ábra. A v sebességkoordináta a z tengely pozitív iránya felől nézve, $Re = 200$

A 12. ábra a kiszámított sebességvektorokat mutatja a rácspontokban. Megfigyelhető, hogy ez az eredeti, nem ekvidisztáns rács. A sebesség nagyságát a vektor nagysága és színe jelzi.



12. ábra. A sebességmező a `quiverc` függvény meghívásával, $Re = 200$

Mind nemlineáris egyenletrendszereknél, mind pedig az időintegrálás során a stabilitás biztosításához folytatásos módszereket szokás alkalmazni. Mi itt a Reynolds-szám, mint paraméter, szerinti folytatást használtuk. Kiszámítunk egy megoldást kisebb Reynolds-számmra, majd a kapott eredményt használjuk fel kezdeti v_0 sebességnek a nagyobb Reynolds-számú eset meghatározásához.

A [17] dolgozat nagy pontosságú megoldásaival hasonlítjuk össze számításainkat. Ott (a

legtöbb cikkhez hasonlóan) a $[0, 1] \times [0, 1]$ négyzetet tekintik, a táblázatokban szereplő koordináták a mi tartományunkhoz vannak igazítva. Az 5-7. táblázatok az üreg közepén mutatják a sebességet.

N	saját számítás, különböző N felosztásokra			[17] dolgozat
	10	20	50	
$u(0, 0)$	-2,05418557e-1	-2,05195217e-1	-2,05191686e-1	-2,05191715e-1
$v(0, 0)$	6,4550503e-6	6,3644848e-6	6,3676966e-6	6,3677058e-6

5. táblázat. A tartomány közepén a sebesség $Re = 0, 01$ esetén

N	saját számítás, különböző N felosztásokra			[17] dolgozat
	10	20	50	
$u(0, 0)$	-2,085100690e-1	-2,091283435e-1	-2,091491343e-1	-2,09149142e-1
$v(0, 0)$	5,7673833e-2	5,7505097e-2	5,7536504e-2	5,7536559e-2

6. táblázat. A tartomány közepén a sebesség $Re = 100$ esetén

N	saját számítás, különböző N felosztásokra			[17] dolgozat
	10	20	50	
$u(0, 0)$	-9,54505227e-2	-1,15344131e-1	-1,15053497e-1	-1,15053628e-1
$v(0, 0)$	6,7025081e-2	5,2218832e-2	5,2058066e-2	5,2058082e-2

7. táblázat. A tartomány közepén a sebesség $Re = 400$ esetén

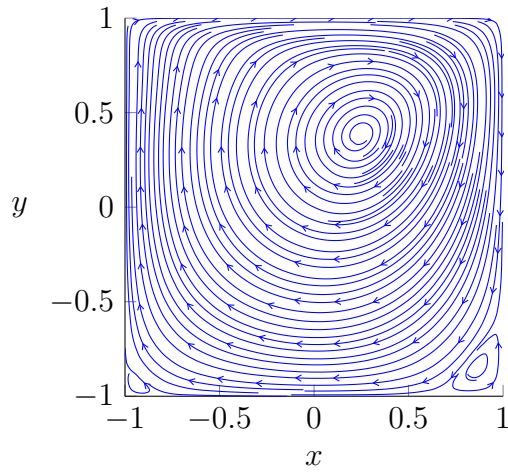
A táblázatokban félkövérrel emeltük ki a normálalakban megegyező tizedesjegyeket. Látható, hogy az u sebességre $N = 50$ -nél minden vizsgált Reynolds-számra hat tizedesjegyre megegyező eredményt kaptunk. A [19] cikkben a sarokponti szingularitásokat eltávolítják, amelynek segítségével jelentősen nő a konvergenciasebesség. Mivel a célunk egy általános megoldó létrehozása volt, ezért mi ezt nem tettük meg.

7.2. Regularizált „lid-driven cavity”

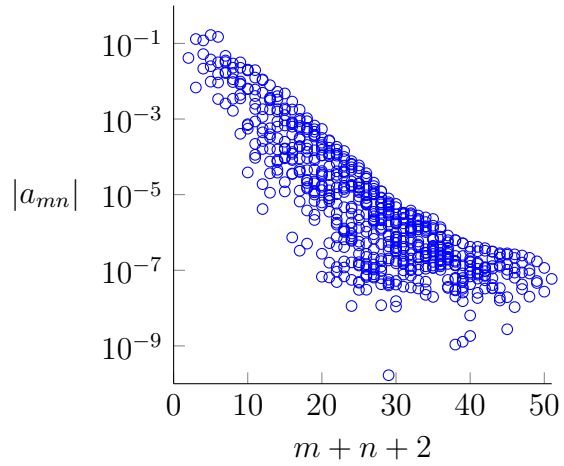
Láttuk, hogy a bal és jobb felső sarka a négyzetnek szinguláris pont. Legyen a mozgó felső lapon a peremfeltétel: $u(x, 1) = (x^2 - 1)^2$. Látható, hogy így már $u(x, y)$ egyértékű az egész peremen. A leállási feltétel mindenhol 10^{-9} relatív eltérésnél volt és $N = 25$ -ös rácson dolgoztunk.

A futási eredményekből kirajzolt áramvonalakat különböző Reynolds-számokra a 13., a 15. és a 17. ábrák tartalmazzák. Látható, hogy Re növelésével az elsődleges örvény középpontja egyre

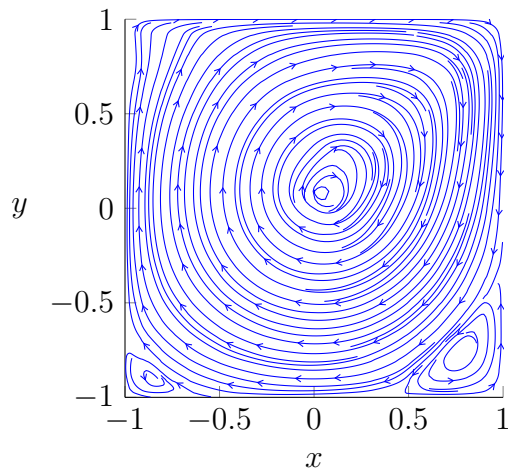
inkább a $(0, 0)$ pont felé tolódik és nő a másodlagos és harmadlagos örvény mérete (l. 13, 15, 17. ábrák).



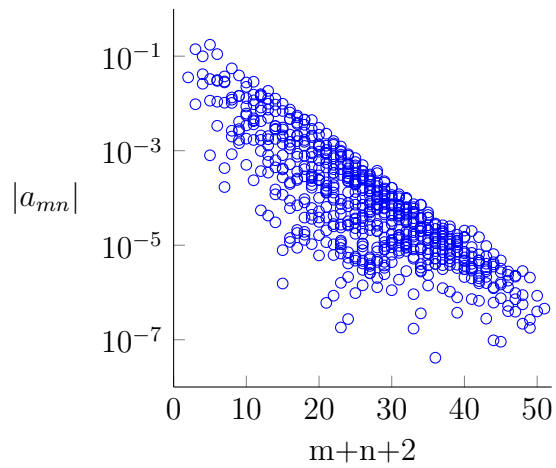
13. ábra. Áramvonalak, $Re = 200$



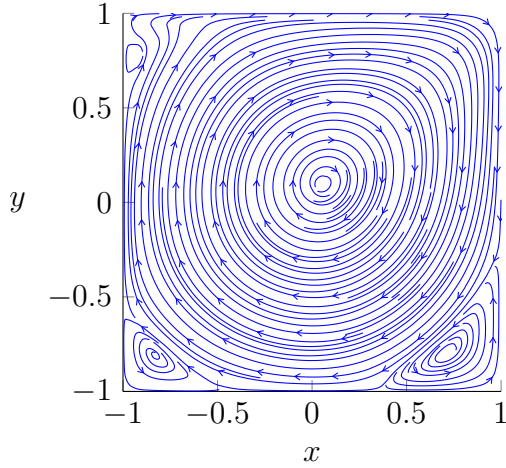
14. ábra. Spektrális együtthatók, $Re = 200$



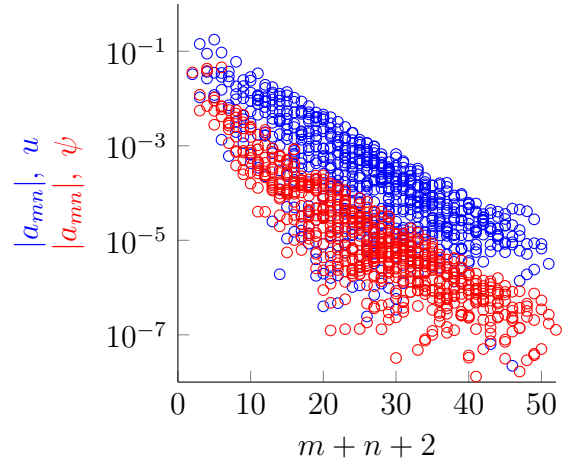
15. ábra. Áramvonalak, $Re = 1000$



16. ábra. Spektrális együtthatók, $Re = 1000$



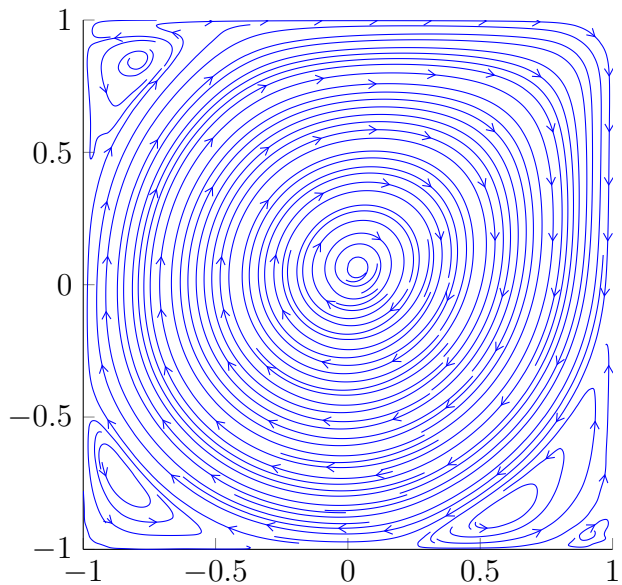
17. ábra. Áramvonalak, $Re = 2000$



18. ábra. Spektrális együtthatók, $Re = 2000$

A spektrális együtthatókat a `chebcoeff` MATLAB függvény segítségével határoztuk meg, majd a `chebcoeffplot`-tal rajzoltattuk ki, amellyel a 14., a 16. és a 18. ábrákat nyertük. Látható, hogy a Reynolds-szám növelésével nő a diszkretizációs hiba. A 14. és 16. ábránál az u sebességre vonatkoznak az együtthatók, míg a 18. ábra tartalmazza a ψ alapváltozóra is. Azt lehet észrevenni, hogy a ψ -nél (és az ω -nál is) körülbelül két nagyságrenddel kisebb az elkövetett hiba.

A Reynolds-szám szerinti folytatást alkalmazva $Re = 8500$ -ig jutottunk. A 19. ábrán látható, hogy az üreg sarkaiban tovább nőttek az örvények. A sorfejtés együtthatóiból kiderült, hogy a megfelelő pontosság (legalább 10^{-5}) eléréséhez nem volt elegendő $N = 25$. Mivel a Reynolds-szám növelésével magasabb fokú közelítés szükséges, továbbá a stabilitás fenntartásához jelentősen kisebb időlépés, ezért nagy Reynolds-szám esetén PC-n a direkt numerikus szimuláció (DNS) túl nagy memória és számítási igénnyel jár. Ekkor turbulencia modelleket használnak.



19. ábra. Spektrális együtthatók, $Re = 8500$

8. Összefoglalás, további célok

A dolgozatban összenyomhatatlan kétdimenziós áramlási jellemzők numerikus megoldásával foglalkoztunk. Az időfüggő Navier-Stokes egyenletet örvényesség-áramfüggvény megfogalmazásban írtuk fel és először időben, majd térben diszkretizáltuk. Szemi-implicit időintegrálást alkalmaztunk, míg térben a nagy pontosságú Chebyshev kollokációs módszert használtuk. A fő célunk egy MATLAB program elkészítése volt. Kétféle időintegrálást implementáltunk, a Navier-Stokes egyenletet kiegészítettük az energiaegyenlettel, így módon hőátadásos feladatokat is meg tudunk oldani. Az eredményekre különböző megjelenítési technikákat alkalmazhatunk és egy stabil interpolációs módszerrel tetszőleges helyen tudjuk előállítani a megoldást. Az örvényvektor és az áramfüggvény bevezetésének előnyeit és hatékonyságát példákon keresztül mutattuk be és összehasonlítottuk a szakirodalomban talált eredményekkel. Az elkészített függvények elég általánosak ahhoz, hogy könnyen bővíthetők legyenek tengelyszimmetrikus feladatokra és spektrálem módszerre is.

A közeljövőben megvalósítandó célunk, hogy a program téglalap tartományt és tengelyszimmetrikus feladatot is kezelni tudjon. A négyzetről téglalap tartományra való általánosítás egy lineáris leképezéssel megoldható, amelyet az úgynevezett eltolt Chebyshev-polinomok valósítanak meg. Hengerszimmetrikus feladatnál – bár a meridiánmetszet független az azimutális szögtől – egy további differenciálegyenletet kapunk az azimutális sebességek koordinátára. Ekkor a feladatot hasonlóan oldjuk meg, mint ahogy azt tettük a konvekció esetén. Eltérés a descartes-i koordináta-rendszerhez képest az, hogy a differenciáloperátorok más alakban állnak elő. A programon belül tehát a két koordináta-rendszer csupán a differenciáló mátrixokban tér el, amely egy *if*-szerkezettel változtatható. Hasznos lenne magasabb rendű időintegrálási módszereket bevezetni, hogy nagyobb pontossággal tudjunk vizsgálni tranziens jelenségeket. Távlati célunk, hogy általános célú spektrálem módszert valósítsunk meg, melynek sok előnye van. Mivel ez egy tartomány felosztásos módszer, ezért komplex – akár többszörösen összefüggő – tartományon is lehet vele dolgozni. További előnye, hogy a szinguláris helyekből eredő konvergencia romlás nem terjed ki az egész tartományra, hanem csak a szinguláris hely egy környezetére. Végül pedig a differenciálegyenletek spektrálem módszerhez használt gyenge alakjában könnyebben előírhatók lesznek a sebességre vonatkozó vegyes peremfeltételek. Érdekes lehet nem-newtoni anyagmodelleket is beépíteni és a komplexitás növekedése miatt objektum orientált stílusra áttérni. A konkrét programozási célok a TODO-ban láthatók.

Köszönetnyilvánítás: A kutatás a TÁMOP 4.2.4.A/2-11-1-2012-0001 azonosító számú Nemzeti Kiválóság Program – Hazai hallgatói, illetve kutatói személyi támogatást biztosító rendszer kidolgozása és működtetése országos program című kiemelt projekt keretében zajlott. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

9. Hivatkozások

- [1] Bangerth, W., Hartmann, R., and Kanschat, G. deal.II – a General Purpose Object Oriented Finite Element Library. *ACM Trans. Math. Softw.*, 33(4):24/1–24/27, 2007. doi: 10.1145/1268776.1268779.
- [2] Berrut, J.-P. and Trefethen, L. N. Barycentric Lagrange Interpolation. *SIAM Rev.*, 46(3): 501–517, 2004.
- [3] Boyd, J. P. *Chebyshev and Fourier Spectral Methods*. Dover Books on Mathematics. Dover Publications, 2000.
- [4] Canuto, C., Hussaini, M. Y., Quarteroni, A., and Zang, T. A. *Spectral Methods in Fluid Dynamics*, volume 16 of *Springer Series in Computational Physics*. Springer Berlin Heidelberg, 1988. doi: 10.1007/978-3-642-84108-8.
- [5] Chorin, A. J. Numerical solution of the Navier-Stokes equations. *Math. Comp.*, 22(6): 745–762, 1968.
- [6] Cox, S. M. and Matthews, P. C. Exponential Time Differencing for Stiff Systems. *J. Comput. Phys.*, 176(2):430–455, 2002. ISSN 0021-9991. doi: <http://dx.doi.org/10.1006/jcph.2002.6995>.
- [7] Csáti, Z. Függőleges körhenger körül kialakuló áramlás jellemzőinek vizsgálata kényszerkonvekció esetén, 2014.
- [8] Daube, O. Resolution of the 2D Navier-Stokes equations in velocity-vorticity form by means of an influence matrix technique. *J. Comput. Phys.*, 103(2):402–414, 1992. ISSN 0021-9991. doi: [http://dx.doi.org/10.1016/0021-9991\(92\)90411-Q](http://dx.doi.org/10.1016/0021-9991(92)90411-Q).
- [9] Deville, M. O., Fischer, P. F., and Mund, E. H. *High-Order Methods for Incompressible Fluid Flow*. Number 9 in Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 1st edition, 2002.
- [10] Horn, R. A. and Johnson, C. R. *Topics in Matrix Analysis*. Cambridge University Press, 1st edition, 1994.
- [11] Horng, T.-L. and Teng, C.-H. An Error Minimized Pseudospectral Penalty Direct Poisson Solver. *J. Comput. Phys.*, 231(6):2498–2509, Mar. 2012. ISSN 0021-9991. doi: 10.1016/j.jcp.2011.11.042.
- [12] Karageorghis, A. A note on the satisfaction of the boundary conditions for Chebyshev collocation methods in rectangular domains. *J. Sci. Comput.*, 6(1):21–26, 1991. ISSN 0885-7474. doi: 10.1007/BF01068122.
- [13] Kopriva, D. A. *Implementing Spectral Methods for Partial Differential Equations: Algorithms for Scientists and Engineers*. Scientific Computation. Springer Netherlands, 2009.

- [14] Lanczos, C. Trigonometric interpolation of empirical and analytical functions. *J. Math. Phys.*, 17:123–199, 1938.
- [15] Li, B. Q. *Discontinuous Finite Elements in Fluid Dynamics and Heat Transfer*, volume 5 of *Computational Fluid and Solid Mechanics*. Springer, 2006.
- [16] Liu, J.-G., Wang, C., and Johnston, H. A Fourth Order Scheme for Incompressible Boussinesq Equations. *J. Sci. Comput.*, 18(2):253–285, Apr. 2003. ISSN 0885-7474. doi: 10.1023/A:1021168924020.
- [17] Marchi, C. H., Suero, R., and Araki, L. The Lid-Driven Square Cavity Flow: Numerical Solution with a 1024×1024 Grid. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 31(3):186–198, 2009.
- [18] Peyret, R. *Spectral Methods for Incompressible Viscous Flow*, volume 148 of *Applied Mathematical Sciences*. Springer, 2001.
- [19] Schultz, W. W., Lee, N. Y., and Boyd, J. P. Chebyshev Pseudospectral Method of Viscous Flows with Corner Singularities. *J. Sci. Comput.*, 4(1):1–24, 1989. ISSN 0885-7474. doi: 10.1007/BF01061264.
- [20] Schumack, M. R., Schultz, W. W., and Boyd, J. P. Spectral Method Solution of the Stokes Equations on Nonstaggered Grids. *J. Comput. Phys.*, 94(1):30–58, 1991. ISSN 0021-9991. doi: [http://dx.doi.org/10.1016/0021-9991\(91\)90136-9](http://dx.doi.org/10.1016/0021-9991(91)90136-9).
- [21] Shen, J. Efficient Spectral-Galerkin Method I. Direct Solvers of Second- and Fourth-Order Equations Using Legendre Polynomials. *SIAM Journal on Scientific Computing*, 15(6): 1489–1505, 1994. doi: 10.1137/0915089.
- [22] Shen, J., Tang, T., and Wang, L.-L. *Spectral Methods: Algorithms, Analysis and Applications*, volume 41 of *Springer Series in Computational Mathematics*. Springer, 2011.
- [23] Weinan, E. and Liu, J.-G. Gauge Method for Viscous Incompressible Flows. *Commun. Math. Sci.*, 1(2):317–332, 2003.

A. MATLAB programok

ABBD2

```
1 function F = ABBD2(Dt,f,B,omega_old,omega_oldold,nu)
2 %SIBE    Second order Adams-Bashforth/backward-difference method for the
3 %Navier-Stokes equation.
4 %
5 %    Inputs: Dt      - time step
6 %              f      - forcing term at the current time
7 %              B      - nonlinear convective term
8 %              omega_old - vorticity one time-step before
9 %              omega_oldold - vorticity two time-steps before
10 %              nu      - kinematic viscosity
11 %
12 %    Output: F - discretized right hand side of the Helmholtz-equation at
13 %              the current time
14 %
15 %    See also    SIBE
16
17 %    Zoltán Csáti
18 %    2014/07/11
19
20 F = 1/nu*(-f + 2*B{end}-B{end-1} - (4*omega_old-omega_oldold)/(2*Dt));
```

barweight

```
1 function weight = barweight(x)
2 %BARWEIGHT    Barycentric weights.
3 %    WEIGHT = BARWEIGHT(X) returns the barycentric weights for an arbitrary
4 %    set of points X.
5 %
6 %    See also    DERMATRIX
7
8 %    The algorithm is based on the article
9 %        Berrut, J.-P. and Trefethen, L. N.: Barycentric Lagrange Interpolation,
10 %        SIAM Rev., vol. 46, no. 3, pp. 501-517, 2004
11 %
12 %    Zoltán Csáti
13 %    2014/05/29
14
15 n = numel(x);
16 x = x(:); % force x to be a column vector
17 X = x(:,ones(1,n)); % same as X = repmat(xj,1,n), but faster
18 sub = X-X.';
```

```

19 sub(logical(eye(n))) = 1; % change the diagonal entries from 0 to 1
20 weight = prod(sub,2); % take the product along the rows
21 weight = 1./weight;

```

baryint2

```

1 function [G X Y] = baryint2(x,F,y)
2 %BARINT    Perform barycentric Lagrange-interpolation in two dimensions.
3 %    [G X Y] = BARINT2(x,F,y) returns the interpolated function values at
4 %    Y based on the known points (X,F) where X (Y) is the tensor product
5 %    grid created from vector x (y), respectively. X and Y are calculated.
6 %    G = BARINT2(x,F,y) returns the interpolated function values at Y
7 %    based on the known points (X,F) where X (Y) is the tensor product
8 %    grid created from vector x (y), respectively. X and Y are not
9 %    calculated.
10 %
11 %    Example: Evaluate the following function on a tensor product grid
12 %              composed by some random x values on [0,1].
13 %              Interpolate the function on [0,1].
14 %              Solution: x = rand(1,10);
15 %                      [X Y] = meshgrid(x);
16 %                      F = @(X,Y) sin(X).*cos(Y);
17 %                      newAbscissas = 0:0.01:1;
18 %                      [interpolated Xnew Ynew] = baryint2(x,F(X,Y), ...
19 %                                                         newAbscissas);
20 %
21 %                      % Compare the interpolated and the analytical values
22 %                      % (the error highly depends on the distribution of the
23 %                      % elements of x):
24 %                      norm(interpolated-F(Xnew,Ynew),Inf);
25 %                      mesh(Xnew,Ynew,interpolated);
26 %
27 %    See also    BARYINT, BARWEIGHT, BARINT
28 %
29 %    The algorithm is based on
30 %    Kopriva D. A.: Implementing Spectral Methods for Partial
31 %    Differential Equations, Springer, 2009
32 %    Zoltán Csáti
33 %    2014/08/04
34
35 % Obtain the barycentric weights
36 w = barweight(x);
37 % Transpose vectors x and y if necessary
38 if size(y,2) == 1 % y is a column vector
39     y = y.';      % y must be a row vector
40 end

```

```

41 x = x(:); % force x to be a column vector
42 % Perform barycentric interpolation
43 numelx = length(x);
44 A = y(ones(numelx,1),:); % equivalently, A = repmat(y,length(x),1);
45 A = bsxfun(@minus,A,x);
46 A = bsxfun(@times,1./A,w);
47 A = A./(repmat(sum(A),numelx,1));
48 % Now the transpose of A is the discrete Lagrange interpolating polynomial.
49 G = transpose(A)*F*A;
50 if nargout > 1
51     % Function meshgrid is a computationally expensive part of the code
52     [X Y] = meshgrid(y);
53 end
54 % If x and y overlap each other for some data points, the formula divides
55 % by zero and the solution in NaN. It does not affect visualization since
56 % MATLAB ignores NaN values.
57 NaNoccurred = isnan(G);
58 if any(NaNoccurred)
59     warning('MATLAB:baryint2:divisionByZero', ...
60         'There are %d overlapping data points.', sum(sum(NaNoccurred)));
61 end

```

chebcoeff

```

1 function B = chebcoeff(A)
2 %CHEBCOEFF Determine spectral coefficients from the nodal values in 2D.
3 % B = CHEBCOEFF(A) determines the spectral coefficients in matrix B
4 % for the 2D Chebyshev expansion using the known nodal values given in
5 % matrix A.
6
7 % The algorithm uses the Fast Chebyshev Transform and the code is
8 % identical to Greg von Winckel's "2D Chebyshev Transform" except the
9 % first line.
10 %
11 % Zoltán Csáti
12 % 2014/09/21
13
14 % Flip the matrix elements because of reverse ordering
15 A = rot90(A,2); % same as flipud(fliplr(A));
16 % From now, use Greg von Winckel's code available from
17 % http://www.mathworks.com/matlabcentral/fileexchange/
18 % 4416-2d-chebyshev-transform
19 [N,M]=size(A);
20 F=ifft([A(1:N,:);A(N+1:-1:2,:)]);
21 B=real([F(1,:); 2*F(2:N,:)]);
22 G=B.';

```

```

23 F=ifft([G(1:M,:);G(M-1:-1:2,:)]);
24 B=real([F(1,:); 2*F(2:M,:)]).';

```

chebcoeffplot

```

1 function chebcoeffplot(A)
2 %CHEBCOEFFPLOT   Logarithm plot of the spectral coefficients.
3 %   CHEBCOEFFPLOT(A) displays the logarithm of the absolute values of the
4 %   spectral coefficients stored in matrix A.
5 %
6 %   See also   CHEBCOEFF
7
8 %   Zoltán Csáti
9 %   2014/09/22
10
11 % We are interested in the absolute value of the coefficients
12 A = abs(A);
13 % Rotate A to get the minor diagonals
14 A = rot90(A,-1);
15 % Initialize figure and axes
16 n = size(A,2);
17 fig = figure('Visible','off');
18 a = axes('Parent',fig, 'YScale','log');
19 xlabel('m+n+2', 'Parent',a);
20 ylabel('|a_{mn}|', 'Parent',a, 'Rotation',0);
21 % Create the "m+n+2 - |a_mn|" plot
22 for k = 1:2*n-1
23     if k <= n % above and on the minor diagonal
24         xvec = (k+1)*ones(k,1);
25     else % under the minor diagonal
26         xvec = (k+1)*ones(2*n-k,1);
27     end
28     line(xvec, diag(A,n-k), 'Parent',a, 'Marker','o', 'LineStyle','none');
29 end
30 % Make the figure visible
31 set(fig, 'Visible','on');

```

convective

```

1 function B = convective(u,v,omega,grid,D1)
2 %CONVECTIVE   Convective term of the Navier-Stokes equation.
3 %
4 %   Input: x      - Chebyshev-Gauss-Lobatto nodes
5 %           u, v  - horizontal and vertical velocity coordinates

```

```

6 %          omega - vorticity
7 %          grid  - meshgrid from x
8 %          D1    - first differentiation matrix
9 %
10 %   Output: B - convective term in vorticity-stream function formulation
11 %
12 %   See also   SIBE, ABBD2, PARTIALDER
13
14 %   Zoltán Csáti
15 %   2014/07/11
16
17 [derx dery] = partialder(omega,grid,D1);
18 B = u.*derx + v.*dery;

```

correctinput

```

1 function varargout = correctinput(x,method,varargin)
2 %CORRECTINPUT   Helper function to handle boundary conditions.
3 %   VARARGOUT = CORRECTINPUT(X,METHOD,VARGIN) processes inputs (numeric
4 %   arrays or function handles), evaluate them at either the boundary
5 %   points without the ends (METHOD='inner') or on the whole boundary
6 %   (METHOD='all') of a side of the square domain.
7 %
8 %   See also
9
10 %   Zoltán Csáti
11 %   2014/08/19
12
13 x = x(:);
14 N = numel(x)-1;
15 method = lower(method);
16 switch method
17     case 'inner'
18         nodes = x(2:N);
19     case 'all'
20         nodes = x;
21     otherwise
22         error('MATLAB:correctinput:incorrectString', ...
23             'Input method must be either ''inner'' or ''all''.');
24 end
25 nroInput = length(varargin);
26 for k = 1:nroInput
27     element = varargin{k}; % current input
28     % Input given as a function handle
29     if isa(element,'function_handle')
30         varargout{k} = element(nodes);

```

```

31     if numel(varargout{k}) == 1
32         varargout{k} = varargout{k}*ones(size(nodes));
33     end
34     % Input given as a numeric array
35     elseif isa(element,'numeric')
36         if numel(element) ~= numel(nodes)
37             error('MATLAB:correctinput:incorrectArray', ...
38                 ['Array length must match the number of boundary nodes ' ...
39                 '(%d).'], numel(nodes));
40         else
41             varargout{k} = element;
42         end
43     else
44         error('MATLAB:correctinput:wrongClass', ['Boundary conditions'...
45             ' must be given as function handles or numeric arrays.']);
46     end
47 end

```

dermatrix

```

1 function D = dermatrix(x,m)
2 %DERMATRIX   Derivative matrices for a set of interpolation points.
3 %   D = DERMATRIX(X,M) returns the first M derivative matrices for a given
4 %   set of interpolation points given as a vector X.
5 %   D = DERMATRIX(X) returns the first order derivative matrix for a given
6 %   set of interpolation points given as a vector X.
7 %
8 %   Example: create the first three derivative matrices for Gauss-Lobatto
9 %   points of order 64
10 %       x = LegGaussLobNodeWeight(64);
11 %       D = dermatrix(x,3);
12 %
13 %   See also   BARWEIGHT, LEGGAUSSLOBNODEWEIGHT, LEGPOLY
14
15 %   The algorithm is based on
16 %       Kopriva D. A.: Implementing Spectral Methods for Partial
17 %       Differential Equations, Springer, 2009
18 %   For matrices of moderate size (n<100) using Legendre, Chebyshev, etc.
19 %   points, the summation is almost free of round-off error. However
20 %   for large matrices or for equidistant points the round-off error can
21 %   be huge. Then use the FFT formulation for differentiation or a stable
22 %   summation algorithm, e.g. Jan Simon's great XSum package:
23 %   http://www.mathworks.com/matlabcentral/fileexchange/26800-xsum
24
25 %   Zoltán Csáti
26 %   2014/06/21

```

```

27
28 x = x(:); % force x to be a column vector
29 n = numel(x);
30 w = barweight(x); % calculate the barycentric weights
31 % Create D in a vectorized way
32 X = x(:,ones(1,n));
33 W = w(:,ones(1,n));
34 D = cell(m,1);
35 D{1} = W'./W.*(1./(X-X'));
36
37 %% Create the first order derivative matrix
38 % Replace the diagonal (which is a vector of Inf or NaN) with zeros
39 D{1}(1:(n+1):end) = 0;
40 % Do the negative sum trick
41 D{1}(1:(n+1):end) = -sum(D{1},2);
42
43 %% Create the higher order derivative matrices using a recursion
44 for k = 2:m
45     diagD = diag(D{k-1});
46     D{k} = k./(X-X').*(W'./W.*diagD(:,ones(1,n))-D{k-1});
47     D{k}(1:(n+1):end) = 0;
48     D{k}(1:(n+1):end) = -sum(D{k},2);
49 end

```

determine_xi

```

1 function xi = determine_xi(N,invM,psi_tilde,hleft,htop,hright, ...
2 hbottom,grid,D1)
3 %DETERMINE_XI    Determines the vector of xi needed for the P_bar-problem.
4 %
5 %    Inputs:  N          - polynomial degree in x and y direction
6 %            invM        - inverse of the influence matrix
7 %            psi_tilde    - stream function solution of the P_tilde-problem
8 %            hleft        - tangential boundary condition on the left edge
9 %            htop         - tangential boundary condition on the top edge
10 %            hright       - tangential boundary condition on the right edge
11 %            hbottom      - tangential boundary condition on the bottom edge
12 %            grid         - meshgrid from the CGL points
13 %            D1           - first differentiation matrix
14 %
15 %    Output:  xi - coefficients obtained from the influence matrix equation
16 %
17 %    See also  PBAR_PROBLEM, PTILDE_PROBLEM, INFLUENCEMATRIX, PARTIALDER
18
19 %    Zoltán Csáti
20 %    2014/07/05

```

```

21
22 %% Construct the right hand side
23 [derx dery] = partialder(psi_tilde,grid,D1);
24 gderleft= -derx(2:N,1);
25 gdertop = dery(end,3:N-1)';
26 gderright = derx(2:N,end);
27 gderbottom = -dery(1,3:N-1)';
28
29 % Discard the four nodes
30 htop(1) = []; htop(end) = [];
31 hbottom(1) = []; hbottom(end) = [];
32 % Change the numbering on the right and on the bottom side
33 hright = hright(end:-1:1);
34 hbottom = hbottom(end:-1:1);
35 % Construct the right hand side vector from the side values
36 RHS = [hleft-gderleft;htop-gdertop;hright-gderright;hbottom-gderbottom];
37 %% Solve the system for xi
38 xi = invM*RHS;

```

diagonalization

```

1 function [P invP Q invQ LambdaX_i LambdaY_j] = diagonalization(N,Dx,Dy)
2 %DIAGONALIZATION Full diagonalization of the Helmholtz-operator.
3 %
4 % Inputs: N - polynomial degree in x and y direction
5 %          Dx - differentiation matrix with respect to x
6 %          Dy - differentiation matrix with respect to y
7 %
8 % Outputs: P - matrix of the eigenvectors of Dx
9 %          Q - matrix of the eigenvectors of Dy
10 %          invP - inverse of P
11 %          invQ - inverse of Q
12 %          LambdaX_i - matrix from the eigenvalues of Dx
13 %          LambdaY_j - matrix from the eigenvalues of Dy
14
15 % Zoltán Csáti
16 % 2014/09/20
17
18 [P LambdaX] = eig(Dx);
19 [Q LambdaY] = eig(Dy);
20 invP = inv(P);
21 invQ = inv(Q);
22 LambdaX_i = diag(LambdaX);
23 LambdaX_i = repmat(LambdaX_i,1,N-1);
24 LambdaY_j = diag(LambdaY)';
25 LambdaY_j = repmat(LambdaY_j,N-1,1);

```

getnode

```
1 function [left top right bottom] = getnode(k,N)
2 %GETNODE    Select current boundary node on 2D spectral grid.
3 %    Inputs:  k - selected node
4 %              N - polynomial degree in x and y direction
5 %
6 %    Outputs: sides of the square
7 %
8 %    The sides are numbered clockwise starting with the left side.
9 %
10 %    See also    PL_PROBLEM
11
12 %    Zoltán Csáti
13 %    2014/07/05
14
15 side = ceil(k/(N-1)); % side of the square
16 localNode = k-(side-1)*(N-1); % node on the current side
17
18 left = zeros(N-1,1);
19 bottom = left;
20 right = left;
21 top = left;
22 switch side % counterclockwise ordering starting with the left side
23     case 1
24         left(localNode) = 1;
25     case 2
26         top(localNode) = 1;
27     case 3
28         right(localNode) = 1;
29         right = right(N-1:-1:1); % reverse ordering
30     case 4
31         bottom(localNode) = 1;
32         bottom = bottom(N-1:-1:1); % reverse ordering
33     otherwise
34         error('MATLAB:getnode:largeInput', ...
35             'Iteration number must be %d at most.', 4*N-4);
36 end
37 % Bound the internal nodes with the corner nodes
38 left = [0; left; 0];
39 bottom = [0; bottom; 0];
40 right = [0; right; 0];
41 top = [0; top; 0];
```

helmholtz_diag

```

1 function [sol X Y] = helmholtz_diag(N,RHS,leftBC,topBC,rightBC, ...
2 bottomBC,sigma,D,grid,diagMatrices)
3 %HELMHOLTZ_DIAG    Solution of the Helmholtz equation on [-1,1]x[-1,1]
4 %with Dirichlet boundary conditions using Chebyshev collocation method
5 %with full diagonalization.
6 %
7 %    Helmholtz equation:  $u_{xx} + u_{yy} - \sigma u = f(x,y)$ 
8 %
9 %    Inputs:  N           - polynomial degree in x and y direction
10 %            RHS          - function of the right-hand side
11 %            leftBC       - boundary condition on the left edge
12 %            topBC        - boundary condition on the top edge
13 %            rightBC      - boundary condition on the right edge
14 %            bottomBC     - boundary condition on the bottom edge
15 %            sigma        - constant
16 %            D            - second order Chebyshev differentiation matrix
17 %            grid         - meshgrid from the CGL points
18 %            diagMatrices - must contain the eigenvectors, their inverses
19 %                        and the eigenvalues as a cell array
20 %
21 %    Outputs: sol - solution at (X,Y)
22 %            X, Y - meshgrid from the Chebyshev-Gauss-Lobatto points
23 %
24 %    The boundary conditions should be given so that the values at the
25 %    corners are the same for the neighbouring boundaries to maintain
26 %    continuity.
27 %
28 %    Used for time-dependent solvers. Examples are provided in the
29 %    standalone version.
30 %
31 %    See also    HELMHOLTZ_DIAG_STANDALONE
32
33 %    The algorithm is based on
34 %        Roger Peyret.: Spectral Methods for Incompressible Viscous Flow,
35 %        Springer, 2002
36 %
37 %    Zoltán Csáti
38 %    2014/09/08
39
40
41 %% Construct the coefficient matrix
42 % Import the computational grid
43 X = grid{1};
44 Y = grid{2};
45
46 %% Construct the right hand side
47 % Evaluate the known function on the right at the nodes
48 X_red = X(2:end-1,2:end-1);

```

```

49 Y_red = Y(2:end-1,2:end-1);
50 if isa(RHS, 'function_handle')
51     b1 = RHS(X_red,Y_red);
52 elseif isa(RHS, 'numeric')
53     b1 = RHS;
54     if numel(RHS) == 1 % perhaps RHS is a constant
55         b1 = b1(ones(N-1));
56     end
57 else
58     error('MATLAB:helmholtz_diag:wrongClass', ...
59         'Input must be either a function handle or a numeric matrix.');
```

```

60 end
61 % Form the internal values of the discretized boundary conditions
62 leftint = leftBC(2:end-1);
63 rightint = rightBC(2:end-1);
64 bottomint = bottomBC(2:end-1);
65 topint = topBC(2:end-1);
66 % Extract the first and last rows and columns from the derivative matrix
67 leftD = D(:,1); leftD = leftD(2:N);
68 rightD = D(:,N+1); rightD = rightD(2:N);
69 % Create the discrete values of the right hand side matrix
70 b2 = repmat(leftD,1,N-1).*repmat(bottomint',N-1,1);
71 b3 = repmat(rightD,1,N-1).*repmat(topint',N-1,1);
72 b4 = repmat(leftD',N-1,1).*repmat(leftint,1,N-1);
73 b5 = repmat(rightD',N-1,1).*repmat(rightint,1,N-1);
74 H = b1-b2-b3-b4-b5;
75
76 %% Perform full diagonalization
77 % Import the required eigenvectors, their inverses and the eigenvalues
78 P = diagMatrices{1};
79 invP = diagMatrices{2};
80 Q = diagMatrices{3};
81 invQ = diagMatrices{4};
82 LambdaX_i = diagMatrices{5};
83 LambdaY_j = diagMatrices{6};
84 % Calculate the inner values
85 H_hat = invP*H*invQ.';
86 U_hat = H_hat./(LambdaX_i+LambdaY_j-sigma);
87 U = P*U_hat*Q.';
88
89 %% Insert the boundary conditions
90 sol = zeros(N+1);
91 sol(2:N,2:N) = U;
92 sol(:,1) = leftBC;
93 sol(:,end) = rightBC;
94 sol(1,:) = bottomBC;
95 sol(end,:) = topBC;

```

helmholtz_kron

```
1 function [sol X Y] = helmholtz_kron(N,RHS,leftBC,topBC,rightBC, ...
2 bottomBC,sigma)
3 %HELMHOLTZ_KRON    Solution of the Helmholtz equation on [-1,1]x[-1,1]
4 %with Dirichlet boundary conditions using Chebyshev collocation method
5 %with Kronecker product.
6 %
7 %    Helmholtz equation:  $u_{xx} + u_{yy} - \sigma u = f(x,y)$ 
8 %
9 %    Inputs:  N          - polynomial degree in x and y direction
10 %            RHS         - function of the right-hand side
11 %            leftBC      - boundary condition on the left edge
12 %            topBC       - boundary condition on the top edge
13 %            rightBC     - boundary condition on the right edge
14 %            bottomBC    - boundary condition on the bottom edge
15 %            sigma       - constant
16 %    Outputs: sol - solution at (X,Y)
17 %            X, Y       - meshgrid from the Chebyshev-Gauss-Lobatto points
18 %    The boundary conditions should be given so that the values at the
19 %    corners are the same for the neighbouring boundaries to maintain
20 %    continuity.
21 %
22 %    Example: Solve  $u_{xx} + u_{yy} = f(x,y)$ ,  $(x,y)$  in  $(-1,1) \times (-1,1)$  with
23 %             $f(x,y) = -8\pi^2 \cos(2\pi x) \sin(2\pi y)$ 
24 %             $u(x,-1) = 0, \quad -1 \leq x \leq 1$ 
25 %             $u(1,y) = \sin(2\pi y), \quad -1 \leq y \leq 1$ 
26 %             $u(x,1) = 0, \quad -1 \leq x \leq 1$ 
27 %             $u(-1,y) = \sin(2\pi y), \quad -1 \leq y \leq 1.$ 
28 %    Solution:
29 %            topBC = @(x) 0; bottomBC = @(x) 0;
30 %            leftBC = @(y) sin(2*pi*y); rightBC = @(y) sin(2*pi*y);
31 %            fcn = @(x,y) -8*pi^2*cos(2*pi*x).*sin(2*pi*y);
32 %            [u X Y] = helmholtz_kron(30,fcn,leftBC,topBC,rightBC,...
33 %                                     bottomBC);
34 %            mesh(X,Y,u);
35 %    Compare it with the exact solution  $u = \cos(2\pi x) \sin(2\pi y)$ 
36 %            u_exact = cos(2*pi*X).*sin(2*pi*Y);
37 %            maxDifference = norm(u-u_exact,Inf); % 4.8621e-14
38 %
39 %    See also    HELMHOLTZ_DIAG, DERMATRIX, CORRECTINPUT
40
41 %    The example is taken from
42 %            Kopriva D. A.: Implementing Spectral Methods for Partial
43 %            Differential Equations, Springer, 2009
44 %
45 %    Zoltán Csáti
```

```

46 % 2014/07/09
47
48 % Form the Chebyshev-Gauss-Lobatto nodes using the exact formula
49 x = -cos(pi*(0:N)/N)';
50 % Compute the derivative matrix
51 D = dermatrix(x,2);
52 D = D{2};
53 % Reduce the size of the coefficient matrix by deleting the first and last
54 % rows and columns
55 D_red = D(2:N,2:N);
56
57 % Construct the coefficient matrix from the strong form
58 I = eye(N-1);
59 A1 = kron(I,D_red);
60 A2 = kron(D_red,I);
61 A3 = sigma*kron(I,I);
62 A = A1 + A2 - A3;
63
64 % Evaluate the known function on the right at the nodes
65 [X Y] = meshgrid(x);
66 X_red = X(2:end-1,2:end-1);
67 Y_red = Y(2:end-1,2:end-1);
68 if isa(RHS, 'function_handle')
69     b1 = RHS(X_red,Y_red);
70 elseif isa(RHS, 'numeric')
71     b1 = RHS;
72     if numel(RHS) == 1 % perhaps RHS is a constant
73         b1 = b1(ones(N-1));
74     end
75 else
76     error('MATLAB:helmholtz_kron:wrongClass', ...
77         'Input must be either a function handle or a numeric matrix.');
```

```

78 end
79 b1 = b1(:); % collapse to a column vector
80 % Evaluate the boundary conditions at the nodes and handle the case when
81 % the given BC is constant
82 [leftBC topBC rightBC bottomBC] = correctinput(x,'all', ...
83 leftBC,topBC,rightBC,bottomBC);
84 % Form the internal values of the discretized boundary conditions
85 leftint = leftBC(2:end-1);
86 rightint = rightBC(2:end-1);
87 bottomint = bottomBC(2:end-1);
88 topint = topBC(2:end-1);
89 % Extract the first and last and columns from the derivative matrix
90 leftD = D(:,1); leftD = leftD(2:N);
91 rightD = D(:,N+1); rightD = rightD(2:N);
92 % Create the discrete values of the right hand side vector
93 b2 = repmat(leftD,1,N-1).*repmat(bottomint',N-1,1);
94 b2 = b2(:);

```

```

95 b3 = repmat(rightD,1,N-1).*repmat(topint',N-1,1);
96 b3 = b3(:);
97 b4 = repmat(leftD',N-1,1).*repmat(leftint,1,N-1);
98 b4 = b4(:);
99 b5 = repmat(rightD',N-1,1).*repmat(rightint,1,N-1);
100 b5 = b5(:);
101 b = b1-b2-b3-b4-b5;
102
103 % Solve the linear system
104 u = A\b;
105 % Reshape the solution so as to be the solution matrix
106 sol = zeros(N+1);
107 sol(2:N,2:N) = reshape(u,N-1,N-1);
108 % Insert the boundary conditions
109 sol(:,1) = leftBC;
110 sol(:,end) = rightBC;
111 sol(1,:) = bottomBC;
112 sol(end,:) = topBC;

```

influencematrix

```

1 function [invM L_red] = influencematrix(N,psi_l,grid,D1)
2 %INFLUENCEMATRIX    Determines the inverse of the influence matrix.
3 %
4 %   Inputs:  N      - polynomial degree
5 %            psi_l  - stream function solution of the P_l-problem
6 %            grid   - meshgrid from the Chebyshev-Gauss-Lobatto points
7 %            D1     - first differentiation matrix
8 %
9 %   Outputs: invM   - inverse of the influence matrix
10 %              L_red - number of boundary points excluding the four corners
11 %                   and four additional points so that the influence
12 %                   matrix is regular
13 %
14 %   See also    PL_PROBLEM, PARTIALDER
15
16 %   Zoltán Csáti
17 %   2014/07/05
18
19 L = size(psi_l,1);
20 L_red = L-4;
21 M = zeros(L_red,L);
22 % Create the influence matrix
23 for l = 1:L
24     [derx dery] = partialder(psi_l{l},grid,D1);
25     % Create the influence matrix and delete the specific points

```

```

26     M(:,1)=[-derx(2:N,1);dery(end,3:N-1)';derx(2:N,end);-dery(1,3:N-1)'];
27 end
28 % Remove the corresponding columns
29 M = [M(:,1:N-1) M(:,N+1:2*N-3) M(:,2*N-1:3*N-3) M(:,3*N-1:4*N-5)];
30 % Invert the influence matrix
31 invM = inv(M);

```

LegGaussLobNodeWeight

```

1 function [node weight] = LegGaussLobNodeWeight(N,tol,maxiter)
2 %LEGGAUSSLOBNODEWEIGHT Legendre-Gauss-Lobatto nodes and weights.
3 % [NODE WEIGHT] = LEGGAUSSLOBNODEWEIGHT(N) calculates the N-th degree
4 % LGL nodes and weights with error tolerance TOL=1e-15 using MAXITER=10
5 % number of iterations.
6 % [NODE WEIGHT] = LEGGAUSSLOBNODEWEIGHT(N,TOL) calculates the N-th
7 % degree LGL nodes and weights with error tolerance TOL using MAXITER=10
8 % number of iterations.
9 % [NODE WEIGHT] = LEGGAUSSLOBNODEWEIGHT(N,TOL,MAXITER) calculates the
10 % N-th degree LGL nodes and weights with error tolerance TOL using
11 % MAXITER number of iterations.
12 %
13 % See also LEGPOLY, DERMATRIX
14
15 % The algorithm is based on
16 % Kopriva D. A.: Implementing Spectral Methods for Partial
17 % Differential Equations, Springer, 2009
18 % Note: a simpler but (for larger N) less stable method uses eigenvalues
19 % instead of the Newton-iteration.
20 %
21 % Zoltán Csáti
22 % 2014/05/30
23
24 % Default termination criteria
25 if nargin < 2
26     tol = 1e-15;
27     maxiter = 10;
28 elseif nargin < 3
29     maxiter = 10;
30 end
31
32 if N==1
33     node = [-1 1];
34     weight = [1 1];
35 else
36     % node(0)=-1 and node(N)=1 and the roots are symmetric to the origin
37     k = (1:(N+1)/2-1)';

```

```

38 % Use an initial approximation for the roots
39 node = - cos((k+0.25)*pi/N - 3/(8*N*pi)*1./(k+0.25));
40 % Perform Newton iteration
41 newtonIter = 1;
42 Delta = 1;
43 while newtonIter <= maxiter && norm(Delta,Inf) > tol*norm(node,Inf)
44     L = legpoly(N,node);
45     q = (2*N+1)/(N+1)*node.*L(:,end) - (2*N+1)/(N+1)*L(:,end-1);
46     qp = (2*N+1)*L(:,end);
47     Delta = -q./qp;
48     node = node + Delta;
49     newtonIter = newtonIter + 1;
50 end
51 isConverged = newtonIter-1<=maxiter && ...
52 norm(Delta,Inf) <= tol*norm(node,Inf);
53 if ~isConverged
54     error(['MATLAB:LegGaussLobNodeWeight:Newton iteration did ',...
55         'not converge with the given tolerance or iteration.']);
56 end
57 % Compute all the N+1 nodes and weights
58 L = legpoly(N,node);
59 weight = 2./(N*(N+1)*L(:,end).^2);
60 node_all = zeros(N+1,1);
61 node_all(1) = -1;
62 node_all(N+1) = 1;
63 node_all(k+1) = node;
64 node_all(N+1-k) = -node;
65 node = node_all;
66 weight_all = zeros(N+1,1);
67 weight_all(1) = 2/(N*(N+1));
68 weight_all(N+1) = 2/(N*(N+1));
69 weight_all(k+1) = weight;
70 weight_all(N+1-k) = weight;
71 weight = weight_all;
72 if mod(N,2) == 0
73     L = legpoly(N,0);
74     weight(N/2+1) = 2./(N*(N+1)*L(:,end).^2);
75     % If N is even, node_N/2 must be zero. However it is automatically
76     % fulfilled as we preallocated node_all to be zeros.
77 end
78 end

```

legpoly

```

1 function [L Lp] = legpoly(N,x)
2 %LEGPOL Legendre polynomials of degree N.

```

```

3 % L = LEGPOLY(N,X) returns the Legendre polynomials and their derivatives
4 % for degree k=0:N at points X.
5 % [L LP] = LEGPOLY(N,X) returns the Legendre polynomials and their
6 % derivatives for degree k=0:N at points X.
7 %
8 % The i-th row and j-th column of L and LP stores  $L_j(x_i)$  and  $L'_j(x_i)$ 
9 % respectively.
10 %
11 % Example: plot the first 5 Legendre polynomials
12 %     x = linspace(-1,1,100);
13 %     L = legpoly(4,x);
14 %     plot(x,L);
15 %
16 % See also LEGENDRE, LEGGAUSSLOBNODEWEIGHT
17
18 % Zoltán Csáti
19 % 2014/05/29
20
21 % Preallocate L and L' matrices and give L_0, L_1, L'_0, L'_1 at x_i
22 L = zeros(length(x),N+1);
23 Lp = L;
24 x = x(:);
25 L(:,1) = ones(length(x),1);
26 L(:,2) = x;
27 Lp(:,2) = L(:,1);
28 % Use the three-term and two-term recurrence relation
29 for k = 2:N
30     L(:,k+1) = (2*k-1)/k*x.*L(:,k) - (k-1)/k*L(:,k-1);
31     Lp(:,k+1) = Lp(:,k-1) + (2*k-1)*L(:,k);
32 end

```

partialder

```

1 function [derx dery Xmesh Ymesh] = partialder(fcn,grid,D)
2 %PARTIALDER First order partial derivatives of a two-variable function.
3 % [DERX DERY XMESH YMESH] = PARTIALDER(FCN,GRID,D) approximates the
4 % first order partial derivatives of the function FCN at the Chebyshev
5 % tensor product grid, GRID using the differentiation matrix D.
6 %
7 % This file is intended to serve in a time-dependent solver. For the
8 % standalone creation of partial derivatives, see the standalone version.
9 %
10 % See also DERMATRIX, PARTIALDER_STANDALONE
11
12 % Zoltán Csáti
13 % 2014/09/20

```

```

14
15
16 Xmesh = grid{1};
17 Ymesh = grid{2};
18 % Put the function to be differentiated into correct form
19 if isa(fcn, 'function_handle')
20     f = fcn(Xmesh, Ymesh);
21 elseif isa(fcn, 'numeric')
22     f = fcn;
23 else
24     error('MATLAB:partialder:wrongClass', ...
25         'Input must be either a function handle or a numeric matrix.');
```

Pbar_problem

```

1 function [omega_bar psi_bar] = Pbar_problem(N, L_red, xi, omega_l, psi_l)
2 %PBAR_PROBLEM    Solves the P_bar-problem for the vorticity and the stream
3 %function.
4 %
5 %    Inputs: N          - polynomial degree
6 %             L_red     - number of boundary points excluding the four corners
7 %                       and four additional points so that the influence
8 %                       matrix is regular
9 %             xi        - coefficients obtained from the influence matrix
10 %                       equation
11 %             omega_l    - solutions of the P_l-problem for the vorticity
12 %             psi_l      - solutions of the P_l-problem for the stream function
13 %
14 %    Outputs: omega_bar - vorticity solution of the P_bar-problem
15 %             psi_bar   - stream function solution of the P_bar-problem
16 %
17 %    See also    PL_PROBLEM
18
19 %    The algorithm is based on
20 %        Roger Peyret.: Spectral Methods for Incompressible Viscous Flow,
21 %        Springer, 2002
22 %
23 %    Zoltán Csáti
24 %    2014/07/05
25
26 % Preallocate the required sizes
27 omega_bar = zeros(size(omega_l{1}));
28 psi_bar = omega_bar;
```

```

29 % Delete the values belonging to the skipped nodes
30 omega_l(N) = []; omega_l(2*N-3) = []; omega_l(3*N-4) = [];
31 omega_l(4*N-7) = [];
32 psi_l(N) = []; psi_l(2*N-3) = []; psi_l(3*N-4) = []; psi_l(4*N-7) = [];
33 % Calculate omega_bar and psi_bar as the sum of the known function values
34 for k = 1:L_red
35     psi_bar = psi_bar + psi_l{k}*xi(k);
36     omega_bar = omega_bar + omega_l{k}*xi(k);
37 end

```

Pl_problem

```

1 function [omega_l psi_l] = Pl_problem(N,sigma,D,grid,diagMatrices)
2 %PL_PROBLEM    Solves the P_l-problem for the vorticity-stream function
3 %formulation of the steady Stokes-equations.
4 %
5 %   Input:   N           - polynomial degree
6 %           sigma        - parameter of the Helmholtz-equation
7 %           D            - second order Chebyshev differentiation matrix
8 %           grid         - meshgrid from the Chebyshev-Gauss-Lobatto pts
9 %           diagMatrices - must contain the eigenvectors, their inverses
10 %                    and the eigenvalues as a cell array
11 %
12 %   Outputs: omega_l - vorticity solution of the P_l problem
13 %            psi_l  - stream function solution of the P_l problem
14 %
15 %   See also   HELMHOLTZ_DIAG, GETNODE, PBAR_PROBLEM
16
17 %   The algorithm is based on
18 %       Roger Peyret.: Spectral Methods for Incompressible Viscous Flow,
19 %       Springer, 2002
20 %
21 %   Zoltán Csáti
22 %   2014/09/08
23
24 L = 4*N-4;
25 omega_l = cell(L,1);
26 psi_l = cell(L,1);
27 zeroBC = zeros(N+1,1);
28 for node = 1:L
29     % Solve the Helmholtz-equation for omega_l
30     RHS = zeroBC(1);
31     [leftBC topBC rightBC bottomBC] = getnode(node,N);
32     omega_l{node} = helmholtz_diag(N,RHS,leftBC,topBC,rightBC,bottomBC,...
33                                     sigma,D,grid,diagMatrices);
34     % Solve the Poisson-equation

```

```

35     RHS = -omega_l{node}(2:N,2:N); % exclude the boundary points
36     psi_l{node} = helmholtz_diag(N,RHS,zeroBC,zeroBC,zeroBC,zeroBC,0, ...
37                                     D,grid,diagMatrices);
38 end

```

Ptilde_problem

```

1 function [omega_tilde psi_tilde X Y] = Ptilde_problem(N,f,gleft,gtop, ...
2 gright,gbottom,sigma,D,grid,diagMatrices)
3 %Solves the P_tilde-problem for the vorticity-stream function formulation
4 %of the steady Stokes-equations.
5 %
6 %   Inputs:  N           - polynomial degree
7 %           f           - external force
8 %           gleft       - normal boundary condition on the left edge
9 %           gtop        - normal boundary condition on the top edge
10 %          gright       - normal boundary condition on the right edge
11 %          gbottom      - normal boundary condition on the bottom edge
12 %          sigma        - parameter of the Helmholtz-equation
13 %          D            - second order Chebyshev differentiation matrix
14 %          grid         - meshgrid from the Chebyshev-Gauss-Lobatto pts
15 %          diagMatrices - must contain the eigenvectors, their inverses
16 %                      and the eigenvalues as a cell array
17 %
18 %   Outputs: omega_tilde - vorticity solution of the P_tilde-problem
19 %           psi_tilde    - stream function solution of the P_tilde-problem
20 %           X, Y        - meshgrid from the Chebyshev-Gauss-Lobatto pts
21 %
22 %   See also  PL_PROBLEM, PBAR_PROBLEM, HELMHOLTZ_DIAG, GETNODE
23
24 %   The algorithm is based on
25 %       Roger Peyret.: Spectral Methods for Incompressible Viscous Flow,
26 %       Springer, 2002
27 %
28 %   Zoltán Csáti
29 %   2014/07/13
30
31 zeroBC = zeros(N+1,1);
32 [omega_tilde X Y] = helmholtz_diag(N,f,zeroBC,zeroBC,zeroBC,zeroBC, ...
33 sigma,D,grid,diagMatrices);
34 psi_tilde = helmholtz_diag(N,-omega_tilde(2:N,2:N),gleft,gtop,gright, ...
35 gbottom,0,D,grid,diagMatrices);

```

showgrid

```

1 function showgrid(N)
2 %SHOWCHEBNODES   Displays the tensor product grid formed by the extrema
3 %of the Chebyshev polynomials of the first kind.
4 %
5 %   See also   SHOWCHEBNODES, CHEB
6
7 %   Zoltán Csáti
8 %   2014/08/06
9
10 if nargin == 0
11     N=10;
12 end
13 x = cos(pi*(0:N)/N);
14 [X Y] = meshgrid(x);
15 figure;
16 plot(X,Y,'ro','MarkerFaceColor','red');

```

showChebNodes

```

1 function showChebNodes(N)
2 %SHOWCHEBNODES   Displays the extrema of the Chebyshev polynomials of the
3 %first kind on the perimeter of the upper curve of a unit circle.
4 %
5 %   See also   SHOWGRID
6
7 %   Zoltán Csáti
8 %   2014/07/03
9
10 if nargin == 0
11     N=10;
12 end
13 xCheb = cos(pi*(0:N)/N);
14 yCheb = sqrt(1-xCheb.^2);
15 x = -1:0.005:1;
16 y = sqrt(1-x.^2);
17 xAxis = zeros(1,N+1);
18 figure; % create a new figure so as not to overwrite the actual one
19 set(gca,'XLim',[-1 1], 'YLim',[0,1.002], 'DataAspectRatio',[1 1 1], ...
20     'NextPlot','Add');
21 line(xCheb,yCheb, 'LineStyle','none', 'Marker','o', 'MarkerSize',8, ...
22     'MarkerFaceColor','blue');
23 line(x,y, 'LineWidth',2);
24 line(xCheb,xAxis, 'LineStyle','none', 'Marker','o', 'MarkerSize',8,...
25     'MarkerFaceColor','red');
26 plot([xCheb; xCheb],[xAxis; yCheb],'b');

```

SIBE

```
1 function F = SIBE(Dt,f,B,omega,nu)
2 %SIBE    Semi-implicit backward-Euler method for the Navier-Stokes equation.
3 %
4 %    Inputs: Dt      - time step
5 %             f      - forcing term at the current time
6 %             B      - nonlinear convective term
7 %             omega  - vorticity one time-step before
8 %             nu     - kinematic viscosity
9 %
10 %    Output: F - discretized right hand side of the Helmholtz-equation at
11 %              the current time
12 %
13 %    See also    ABBD2
14
15 %    Zoltán Csáti
16 %    2014/07/11
17
18 F = 1/nu*(-f + B{end} - omega/Dt);
```

solveNS_ABBD2

```
1 function [omega psi u v X Y k] = solveNS_ABBD2(N,f,gleft,gtop,gright, ...
2 gbottom,hleft,htop,hright,hbottom,u0,v0,Dt,nu,maxIter,reltol)
3 %SOLVENS_ABBD2 Solve the unsteady Navier-Stokes-problem on [-1,1]x[-1,1]
4 %with semi-implicit second order Adams-Bashforth/Backward Differentiation
5 %temporal discretization and Chebyshev spectral collocation in space.
6 %
7 %    Inputs:  N      - polynomial degree in x and y direction
8 %             f      - external force
9 %             gleft  - normal boundary condition on the left edge
10 %             gtop   - normal boundary condition on the top edge
11 %             gright - normal boundary condition on the right edge
12 %             gbottom - normal boundary condition on the bottom edge
13 %             hleft  - tangential boundary condition on the left edge
14 %             htop   - tangential boundary condition on the top edge
15 %             hright - tangential boundary condition on the right edge
16 %             hbottom - tangential boundary condition on the bottom edge
17 %             nu     - kinematic viscosity
18 %             maxIter - maximum number of time iterations
19 %             u0, v0 - initial condition for the field of velocity
20 %                   components
21 %             Dt     - time-step
22 %             nu     - kinematic viscosity
```

```

23 %          maxIter - maximum number of time iterations
24 %          reltol  - relative difference between the n-th and the
25 %                   n-1-th u function values in L2 norm
26 %
27 %   Outputs: X,Y    - meshgrid from the Chebyshev-Gauss-Lobatto points
28 %               omega - solution of omega(x,y) at (X,Y) at t=k*Dt
29 %               psi   - solution of psi(x,y) at (X,Y) at t=k*Dt
30 %               u, v  - solution of u(x,y) and v(x,y) at (X,Y) at t=k*Dt,
31 %                   k = 1,...,min(maxIter,stop) and stop is an
32 %                   appropriate termination criterium
33 %               stat  - statistics about the cause of termination, the number
34 %                   of iterations and the relative error
35 %
36 %
37 %   See also   SOLVENS_SIBE, SOLVESTOKES, PL_PROBLEM, PTILDE_PROBLEM,
38 %               PBAR_PROBLEM, INFLUENCEMATRIX, DETERMINE_XI, DERMATRIX,
39 %               PARTIALDER, DIAGONALIZATION
40
41 %   Zoltán Csáti
42 %   2014/10/19
43
44
45 %% Preprocessing
46 % Create the Chebyshev-nodes (ascending order)
47 x = -cos(pi*(0:N)/N);
48 % Create the tensor product mesh
49 [X Y] = meshgrid(x);
50 grid = {X,Y};
51 % Build the differentiation matrices
52 D = dermatrix(x,2);
53 D1 = D{1};
54 D2 = D{2};
55 % Interpret and evaluate boundary values
56 [gleft gtop gright gbottom] = correctinput(x,'all', ...
57                                           gleft,gtop,gright,gbottom);
58 [hleft htop hright hbottom] = correctinput(x,'inner', ...
59                                           hleft,htop,hright,hbottom);
60 % Diagonalize the Helmholtz-operator
61 [P invP Q invQ LambdaX_i LambdaY_j] = diagonalization(N,D2(2:N,2:N), ...
62                                                         D2(2:N,2:N));
63 diagMatrices = {P,invP,Q,invQ,LambdaX_i,LambdaY_j};
64 % Solve the Pl-problem
65 sigma = 3/(2*nu*Dt);
66 [omega_l psi_l] = Pl_problem(N,sigma,D2,grid,diagMatrices);
67 % Invert the influence matrix
68 [invM L_red] = influencematrix(N,psi_l,grid,D1);
69
70
71 %% Solve at each time-cycle

```

```

72 % Import results from the first time-step using SIBE
73 [omega psi u v X Y k f omega0 B0] = solveNS_SIBE(N,f,gleft,gtop,gright,...
74         gbottom,hleft,htop,hright,hbottom,u0,v0,Dt,nu,1,1e-9);
75 % Preallocate arrays
76 omega{maxIter} = [];
77 psi{maxIter} = [];
78 u{maxIter} = [];
79 v{maxIter} = [];
80 omega_bar = omega;
81 psi_bar = omega;
82 omega_tilde = omega;
83 psi_tilde = omega;
84 B = cell(2,1); % convective term
85 B{2} = B0;
86 % Temporal discretization
87 for k = 2:maxIter
88     B{1} = B{2};
89     B{2} = convective(u{k-1},v{k-1},omega{k-1},grid,D1);
90     B{2} = B{2}(2:N,2:N);
91     if k == 2
92         F = ABBD2(Dt,f,B,omega{k-1}(2:N,2:N),omega0(2:N,2:N),nu);
93     else
94         F = ABBD2(Dt,f,B,omega{k-1}(2:N,2:N),omega{k-2}(2:N,2:N),nu);
95     end
96     % Solve the P_tilde-problem
97     [omega_tilde{k} psi_tilde{k}] = Ptilde_problem(N,F,gleft,gtop, ...
98         gright,gbottom,sigma,D2,grid,diagMatrices);
99     % Solve the influence matrix equation
100    xi = determine_xi(N,invM,psi_tilde{k},hleft,htop,hright,hbottom, ...
101        grid,D1);
102    % Use the above result to construct the P_bar-solution
103    [omega_bar{k} psi_bar{k}] = Pbar_problem(N,L_red,xi,omega_l,psi_l);
104    % Add the P_tilde solution and the P_bar solution which solves
105    % the P-problem
106    omega{k} = omega_tilde{k} + omega_bar{k};
107    psi{k} = psi_tilde{k} + psi_bar{k};
108    % Determine omega uniquely on the boundary
109    omega{k} = -(psi{k}*D2' + D2*psi{k});
110    % Determine the velocity coordinates using the definition of
111    % stream function
112    [xder yder] = partialder(psi{k},grid,D1);
113    u{k} = yder; v{k} = -xder;
114    if k > 2
115        if isnan(norm(u{k}-u{k-1},Inf))
116            difference = NaN;
117            % Return statistics
118            stat.terminationCause = 'Did not converge.';
119            stat.iter = k;
120            stat.error = difference;

```

```

121         break
122     else
123         difference = norm(u{k}-u{k-1},2)/norm(u{k},2);
124         if difference < reltol
125             % Return statistics
126             stat.terminationCause = 'Desired error tolerance reached.';
127             stat.iter = k;
128             stat.error = difference;
129             break
130         end
131     end
132 end
133 end
134
135 % Delete empty cells (a very fast method based on http://www.mathworks.com/
136 % matlabcentral/answers/27042-how-to-remove-empty-cell-array-contents
137 notEmpty = ~cellfun('isempty',omega);
138 omega = omega(notEmpty);
139 psi = psi(notEmpty);
140 u = u(notEmpty);
141 v = v(notEmpty);
142 if k == maxIter
143     stat.terminationCause = 'Maximum iteration number reached.';
144     stat.iter = k;
145     stat.error = difference;
146 end

```

solveNS_SIBE.m

```

1
2 function [omega psi u v X Y k f omega0 B0] = solveNS_SIBE(N,f,gleft, ...
3 gtop,gright,gbottom,hleft,htop,hright,hbottom,u0,v0,Dt,nu,maxIter,reltol)
4 %SOLVENS_SIBE Solve the unsteady Navier-Stokes-problem on [-1,1]x[-1,1]
5 %with semi-implicit backward-Euler temporal discretization and Chebyshev
6 %spectral collocation in space.
7 %
8 % Inputs:  N      - polynomial degree in x and y direction
9 %          f      - external force
10 %          gleft  - normal boundary condition on the left edge
11 %          gtop   - normal boundary condition on the top edge
12 %          gright - normal boundary condition on the right edge
13 %          gbottom - normal boundary condition on the bottom edge
14 %          hleft  - tangential boundary condition on the left edge
15 %          htop   - tangential boundary condition on the top edge
16 %          hright - tangential boundary condition on the right edge
17 %          hbottom - tangential boundary condition on the bottom edge

```

```

18 %          u0, v0 - initial condition for the field of velocity
19 %                      components
20 %          Dt      - time-step
21 %          nu      - kinematic viscosity
22 %          maxIter - maximum number of time iterations
23 %          reltol  - relative difference between the n-th and the
24 %                      n-1-th u function values in L2 norm
25 %
26 %  Outputs: X,Y      - meshgrid from the Chebyshev-Gauss-Lobatto points
27 %          omega     - solution of omega(x,y) at (X,Y) at t=k*Dt
28 %          psi       - solution of psi(x,y) at (X,Y) at t=k*Dt
29 %          u, v      - solution of u(x,y) and v(x,y) at (X,Y) at t=k*Dt,
30 %                      k = 1,...,min(maxIter,stop) and stop is an
31 %                      appropriate termination criterium
32 %          stat      - statistics about the cause of termination, the
33 %                      number of iterations and the relative error
34 %          f         - external force
35 %          omega0    - initial vorticity field
36 %          B0       - initial convective term
37 %
38 %  See also  SOLVENS_ABBD2, SOLVESTOKES, PL_PROBLEM, PTILDE_PROBLEM,
39 %             PBAR_PROBLEM, INFLUENCEMATRIX, DETERMINE_XI, DERMATRIX,
40 %             PARTIALDER, DIAGONALIZATION
41
42 %  Zoltán Csáti
43 %  2014/10/19
44
45
46
47 %% Preprocessing
48 % Create the Chebyshev-nodes (ascending order)
49 x = -cos(pi*(0:N)/N)';
50 % Create the tensor product mesh
51 [X Y] = meshgrid(x);
52 grid = {X,Y};
53 % Build the differentiation matrices
54 D = dermatrix(x,2);
55 D1 = D{1};
56 D2 = D{2};
57 % Interpret and evaluate boundary values
58 [gleft gtop gright gbottom] = correctinput(x,'all', ...
59                                           gleft,gtop,gright,gbottom);
60 [hleft htop hright hbottom] = correctinput(x,'inner', ...
61                                           hleft,htop,hright,hbottom);
62 % Diagonalize the Helmholtz-operator
63 [P invP Q invQ LambdaX_i LambdaY_j] = diagonalization(N,D2(2:N,2:N), ...
64 D2(2:N,2:N));
65 diagMatrices = {P,invP,Q,invQ,LambdaX_i,LambdaY_j};
66 % Solve the Pl-problem

```

```

67 sigma = 1/(nu*Dt);
68 [omega_l psi_l] = Pl_problem(N,sigma,D2,grid,diagMatrices);
69 % Invert the influence matrix
70 [invM L_red] = influencematrix(N,psi_l,grid,D1);
71
72
73 %% Solve at each time-cycle
74 % Preallocate arrays
75 omega = cell(maxIter,1);
76 psi = omega;
77 u = omega;
78 v = omega;
79 omega_bar = omega;
80 psi_bar = omega;
81 omega_tilde = omega;
82 psi_tilde = omega;
83 % Evaluate the known solution at t=0
84 X_red = X(2:N,2:N);
85 Y_red = Y(2:N,2:N);
86 if isa(f,'function_handle')
87     f = f(X_red,Y_red);
88     if numel(f) == 1 % perhaps RHS is a constant
89         f = f(ones(N-1));
90     end
91 elseif isa(f,'numeric')
92     if numel(f) == 1 % perhaps RHS is a constant
93         f = f(ones(N-1));
94     end
95 end
96 u{1} = u0;
97 v{1} = v0;
98
99 % Temporal discretization
100 for k = 1:maxIter
101     if k==1 % initialial time step
102         [~,dery_u] = partialder(u0,grid,D1);
103         [derx_v,~] = partialder(v0,grid,D1);
104         omega0 = derx_v - dery_u;
105         omega{1} = omega0;
106         % Evaluate the convective term
107         B = convective(u0,v0,omega0,grid,D1);
108         B = B(2:N,2:N);
109         B0 = B;
110         F = SIBE(Dt,f,{B},omega0(2:N,2:N),nu);
111         difference = [];
112     else
113         B = convective(u{k-1},v{k-1},omega{k-1},grid,D1);
114         B = B(2:N,2:N);
115         F = SIBE(Dt,f,{B},omega{k-1}(2:N,2:N),nu);

```

```

116     end
117     % Solve the P_tilde-problem
118     [omega_tilde{k} psi_tilde{k}] = Ptilde_problem(N,F,gleft,gtop, ...
119     gright,gbottom,sigma,D2,grid,diagMatrices);
120     % Solve the influence matrix equation
121     xi = determine_xi(N,invM,psi_tilde{k},hleft,htop,hright,hbottom,...
122     grid,D1);
123     % Use the above result to construct the P_bar-solution
124     [omega_bar{k} psi_bar{k}] = Pbar_problem(N,L_red,xi,omega_l,psi_l);
125     % Add the P_tilde solution and the P_bar solution which solves
126     % the P-problem
127     omega{k} = omega_tilde{k} + omega_bar{k};
128     psi{k} = psi_tilde{k} + psi_bar{k};
129     % Determine omega uniquely on the boundary
130     omega{k} = -(psi{k}*D2' + D2*psi{k});
131     % Determine the velocity coordinates using the definition of
132     % stream function
133     [xder yder] = partialder(psi{k},grid,D1);
134     u{k} = yder; v{k} = -xder;
135     if k > 1
136         if isnan(norm(u{k}-u{k-1},Inf))
137             difference = NaN;
138             % Return statistics
139             stat.terminationCause = 'Did not converge.';
140             stat.iter = k;
141             stat.error = difference
142             break
143         else
144             % Maximum difference in the horizontal velocity between the
145             % k-1-th and k-th timestep
146             difference = norm(u{k}-u{k-1},2)/norm(u{k},2);
147             if difference < reltol
148                 % Return statistics
149                 stat.terminationCause = 'Desired error tolerance reached.';
150                 stat.iter = k;
151                 stat.error = difference;
152                 break
153             end
154         end
155     end
156 end
157
158 % Maximum difference in the horizontal velocity between the k-1-th and
159 % k-th timestep
160 if k > 1
161     disp(difference);
162 end
163 % Delete empty cells (a very fast method based on http://www.mathworks.
164 % com/matlabcentral/answers/27042-how-to-remove-empty-cell-array-contents

```

```

165 notEmpty = ~cellfun('isempty', omega);
166 omega = omega(notEmpty);
167 psi = psi(notEmpty);
168 u = u(notEmpty);
169 v = v(notEmpty);
170 if k == maxIter
171     stat.terminationCause = 'Maximum iteration number reached.';
172     stat.iter = k;
173     stat.error = difference;
174 end

```

TODO

```

1 %TODO Workflow I should make to complete the project
2
3 % - Make an install script (or function)
4 % - Compute the spectral coefficients to estimate the error (completed)
5 % - Use the method of manufactured solution to estimate error
6 % - During diagonalization, compute the eigenvalues only once (completed)
7 % - Do not compute the derivative matrix, only once (completed)
8 % - Compute omega on the boundary (completed)
9 % - Other temporal discretization schemes
10 % - Adaptive time-stepping
11 % - Extend the Stokes solver to Navier-Stokes solver
12 %   - time-independent solver
13 %   - time-dependent solver (completed)
14 % - Include temperature (completed)
15 % - Determine pressure if needed
16 % - Use different constitutive laws
17 % - Possibility of various boundary conditions
18 %   - outflow boundary condition
19 %   - wall, moving wall (completed)
20 %   - velocity inlet, etc. (completed)
21 % - Handle boundary conditions in both function handle and a numeric array
22 %   (completed)
23 % - Create a structure that hold the boundary conditions and the options
24 %   (like the odeset/odeget pair)
25 % - Overwrite the date of creation when functions are updated
26 % - Improve helps
27 % - Utility functions
28 %   - show tensor product mesh (completed)
29 %   - show the Chebyshev points on a unit circle (completed)
30 %   - show iteration on progress bar
31 %   - helper functions
32 % - Write documentation
33 % - Eliminate Kronecker product wherever possible (completed)

```

```

34 % - Visualization tools
35 %   - mesh, surface, vector, streamline plots (completed)
36 %   - error plot (log scale)
37 %   - animation for time-dependent case
38 % - Use the more stable barycentric interpolation instead of interp2
39 %   (completed)
40 % - Fast Poisson-solver
41 %   - use the weak form with Cholesky decomposition
42 %   - diagonalization (completed)
43 %   - iterative method using a non-equidistant FDM preconditioner
44 %   - make use of sparse matrices
45 % - FFT-based creation of spectral differentiation matrices
46 % - Use LU decomposition instead of inversion wherever possible
47 % - Iterative solver
48 %   - preconditioning
49 %   - spectral multigrid
50 % - Profiling for better speed (completed)
51 % - Less function overhead (completed)
52 % - Error estimation
53 %   - a posteriori estimation (completed)
54 %   - make an adaptive solver
55 % - Allow axisymmetric problems
56 % - Extension to rectangular domain
57 % - Execution on the GPU (not worth)
58 % - Use mex files if necessary
59 % - Turn to object oriented style
60 % - Make a GUI

```

visualizeResults

```

1 function visualizeResults(varargin)
2 %VISUALIZERESULTS   Visualize data on 2D grids.
3 %   Required input arguments (in this order):
4 %       X - grid data from the x coordinates (MxN matrix)
5 %       Y - grid data from the y coordinates (MxN matrix)
6 %       u - function values at (X,Y)
7 %   Optional input argument
8 %       v - function values at (X,Y)
9 %       default: [];
10 %   Parameter-value pair
11 %       'method' - one of 'quiver','quiverc','mesh','surf','contour',
12 %                   'streamslice'
13 %       default: 'quiver'
14 %       'quiverc' can be downloaded from the File Exchange
15 %       'arrowlength' - one of 'equal' or 'default'
16 %       default: 'default'

```

```

17 %           If 'equal' is given, the length of the arrows in quiver are
18 %           equal. Using 'default', the default length scaling is applied.
19 %
20 %   See also   QUIVER, QUIVERC, STREAMSLICE, CONTOUR, MESH, SURF
21
22 %   Zoltán Csáti
23 %   2014/09/15
24
25 % Handle inputs
26 p = inputParser;
27 sizeX = size(varargin{1});
28 validSize = @(x) validateattributes(x,{'numeric'}, {'size',sizeX});
29 validMethods = @(x) any(strcmpi(x,{'quiver','quiverc','mesh','surf', ...
30 'contour','streamslice'}));
31 validArrowLength = @(x) any(strcmpi(x, {'equal','default'}));
32 p.addRequired('X', @ismatrix);
33 p.addRequired('Y', validSize);
34 p.addRequired('u', validSize);
35 p.addOptional('v', [], validSize);
36 p.addParamValue('method', 'quiver', validMethods);
37 p.addParamValue('arrowlength', 'default', validArrowLength);
38 p.parse(varargin{:});
39 X = p.Results.X;
40 Y = p.Results.Y;
41 u = p.Results.u;
42 v = p.Results.v;
43 method = p.Results.method;
44 arrowLength = p.Results.arrowlength;
45 if isempty(v) ... % v is not supplied
46     && any(strcmp(method,{'quiver','quiverc','streamslice'}))
47         error('MATLAB:visualizeResults:notEnoughInputArguments', ...
48             'Four matrices are required for the specified method.');
```

```

49 end
50 % Plot data
51 method = lower(method);
52 figure;
53 if ~strcmp(method,'quiver') && numel(varargin)>6 % arrowlength prop given
54     warning('MATLAB:visualizeResults:ignoredInput', ...
55         'Arrowlength is taken into account only in case of quiver plot.');
```

```

56 end
57 switch method
58     case 'quiver'
59         if strcmpi(arrowLength,'equal')
60             L = sqrt(u.^2+v.^2);
61             quiver(X,Y,u./L,v./L);
62         else
63             quiver(X,Y,u,v);
64         end
65     case 'quiverc'
```

```

66     if max(sizeX) > 50
67         error('MATLAB:visualizeResults:largeInput', ...
68             ['For computer responsiveness, maximum size is ', ...
69             'limited to 50. Use quiver for larger input.']);
70     end
71     quiverc(X,Y,u,v);
72     case 'streamslice'
73         streamslice(X,Y,u,v,2); % double density
74     case 'contour'
75         contour(X,Y,u);
76     case 'mesh'
77         mesh(X,Y,u);
78     case 'surf'
79         surf(X,Y,u); shading('interp');
80 end
81 % Axes settings
82 axis square; set(gca,'XLim',[-1 1], 'YLim',[-1,1]);

```