VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Magistro baigiamasis darbas

Nestacionaraus skysčio tekėjimo modeliavimas plonų vamzdelių struktūrose

Non-steady Fluid Flow Modelling in Thin Tube Structure

Justas Jankūnas

VILNIUS 2016

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

KATEDRA

Darbo vadovas Docentė, Dr. Olga Štikonienė

Darbo recenzentas _____

Darbas apgintas _____

Darbas įvertintas _____

Registravimo NR. _____

Santrauka

Nestacionaraus skysčio tekėjimo modeliavimas plonų vamzdelių struktūrose

Santrauka

Magistro baigiamajame darbe nagrinėjami skysčio tekėjimo uždavinių skaitinio sprendimo metodai OpenFOAM programine įranga. Apžvelgiamas uždavinio sprendimo ciklas: geometrijos paruošimas, tinklo sugeneravimas, uždavinio apibrėžimas ir sprendimas, rezultatų apdorojimas. Antroje darbo dalyje atliekami eksperimentai ir nagrinėjami jų rezultatai su „Y“ formos kontūrais, keičiant įvairius kontūro ir uždavinio parametrus (bifurkacijos kampo dydis, tinklo elementų dydis, pradinis greitis), siekiant įvertinti slėgio šuolių dydį ir priežastis.

Raktiniai žodžiai : Navjė-Stokso lygtys, skaitiniai metodai, transporto lygtys, skaitinis sprendimas, OpenFOAM, skysčio tekėjimas

Abstract

Non-steady Fluid Flow Modelling in Thin Tube Structure

Abstract

This thesis analyse methods of solving Computational fluid dynamics problems with open-source OpenFOAM software. Full lifecycle of defining, solving and postprocessing computational fluid dynamics problem is presented. In the second part of the thesis, geometries of "Y" type are used to experiment with various parameters (size of bifurcation angle, mesh element size, inlet velocity) to research pressure hikes in the bifurcation point.

Key words : Computational fluid dynamis, CFD, Navier Stokes, OpenFOAM, pressure hikes

Turinys

1 skyrius

Įvadas

1.1 Įvadas

Pirmosios nespūdaus skysčio tekėjimo lygtys buvo suformuotos 1921 metais prancūzų inžinieriaus K. Navjė, o 1845 metais patikslintos anglų matematiko G. Stokso. Dėl šios priežasties skysčių tekėjimo lygtys dažnai yra vadinamos Navjė-Stokso lygtimis. Navjė-Stokso lygtys yra plačiai nagrinėjamos šiuolaikinėje matematikoje, o mokslininkai matematikai bando įrodyti nestacionarios Navjė-Stokso lyčių sistemos išsprendžiamumą, jos sprendinių glodumą ir vienatį. Vilniaus universiteto matematikos ir informatikos fakultetas stipriai prisideda prie Navjė-Stokso lygčių analizinio ir skaitinio (eksperimentinio) tyrinėjimo. K. Pilecko ir G. Panasenko darbe [9] nagrinėjamas Navjė-Stokso uždavinių sprendimas asimptotiniais metodais kontūruose sudarytuose iš plonų vamzdelinių tarpusavyje sujungtų struktūrų. Straipsnyje sukonstruojamas sprendinio asimptotinis sprendinys, pateikiamas paklaidos įvertis tarp tikslaus ir asimptotinio sprendinio.

Strapsnyje nagrinėjama nestacionari Navjė-Stokso lygtis:

$$\begin{cases} \mathbf{v}_t - \nu\Delta\mathbf{v} + (\mathbf{v}\cdot\nabla)\mathbf{v} + \nabla p = 0, \\ div\ \mathbf{v} = 0, \\ \mathbf{v}|_{\partial B_\varepsilon} = g, \mathbf{v}(x,0) = 0. \end{cases} \tag{1.1}$$

ant plonų vamzdelių struktūros B_ε:

$$B_\varepsilon = \left(\cup_{j=1}^{N} \Pi_\varepsilon^{e_j} \right) \cup \left(\cup_{j=1}^{N_1} \omega_\varepsilon^j \right) \tag{1.2}$$

Čia $\Pi_\varepsilon^{e_j}$ yra cilindras, kurio skersmuo ε, o e_j grafo einančio per cilindro centrą kraštinė:

$$\Pi_\varepsilon^{e_j} = \{x^{e_j} \in \mathbb{R}^n : x_n^{e_j} \in (0, |e_j|), \frac{x^{(e_j)'}}{\varepsilon} \in \sigma^{e_j}\} \tag{1.3}$$

čia $x^{e_j} = (x_1^{e_j}, ..., x_{n-1}^{e_j})$, e_j - j-osios grafo kraštinės ilgis (vamzdelio aukštis). ω_ε^j - grafo susikirtimo taško O_j aplinka, t.y.:

$$\omega_\varepsilon^j = \{x \in \mathbb{R} : \frac{x - O_j}{\varepsilon} \in \omega^j\} \tag{1.4}$$

Čia ω^j - nepriklausančios nuo ε vamzdelių susikirtimo grafo viršūnėje O^j sritys.

[9] straipsnyje suformuluojamas asimptotinis 1.1 uždavinio sprendinys kontūro B_ε atveju. Greičio v asimptotinis įvertis:

$$\begin{aligned} v^{(J)}(x,t) = &\sum_{i=1}^{M} \zeta\Big(\frac{x_n^{e_i}}{3r\varepsilon}\Big)\zeta\Big(\frac{|e| - x_n^{(e_j)}}{3r\varepsilon}\Big)\mathbf{V}^{[e_i,J]}\Big(\frac{x^{(e_i)'}}{\varepsilon}, t\Big) \\ &+\sum_{l=1}^{N}\Big(1 - \zeta\Big(\frac{|x - O_l|}{|e|_{min}}\Big)\Big)\mathbf{V}^{[BLO_l,J]}\Big(\frac{x - O_l}{\varepsilon}, t\Big), \end{aligned} \tag{1.5}$$

čia

$$\begin{aligned} \mathbf{V}^{[e,J]} &= (P^{(e)})^*(0, ..., 0, \tilde{V}^{[e,J]})^* \\ \mathbf{V}_{(j)}^{(e)} &= (P^{(e)})^*(0, ..., 0, \tilde{V}_{(j)}^{(e)})^* \\ \tilde{V}^{[e,J]}(y^{(e)'}, t) &= \sum_{j=0}^{J} \varepsilon^j \tilde{V}_{(j)}^{(e)}(y^{(e)'}, t) \\ \mathbf{V}^{[BLO_i,J]}(y,t) &= \sum_{j=0}^{J} \varepsilon^j \tilde{\mathbf{V}}_{(j)}^{[BLO_i]}(y,t) \end{aligned} \tag{1.6}$$

Slėgis p asimptotiškai apskaičiuojamas taip:

$$\begin{aligned} p^{(j)}(x,t) = &\sum_{i=1}^{M} \zeta\Big(\frac{x_n^{e_i}}{3r\varepsilon}\Big)\zeta\Big(\frac{|e| - x_n^{(e_j)}}{3r\varepsilon}\Big)(-s^{(e_i)}(t)x_n^{(e_i)} + a^{e_i}(t)) \\ &+\frac{1}{\varepsilon}\sum_{l=0}^{N}\Big(1 - \zeta\Big(\frac{|x - O_l|}{|e|_{min}}\Big)\Big)P^{[BLO_l,J]}\Big(\frac{x - O_l}{\varepsilon}, t\Big), \end{aligned} \tag{1.7}$$

kur

$$\begin{aligned} s^{(e)}(t) &= \frac{1}{\varepsilon^2}\sum_{j=0}^{J} \varepsilon^j s_j^{(e)}(t), \\ a^{(e)}(t) &= \frac{1}{\varepsilon^2}\sum_{j=0}^{J} \varepsilon^j a_j^{(e)}(t), \\ P^{[BLO_l,J]}(y,t) &= \sum_{j=-1}^{J} \varepsilon^j P_j^{[BLO^l,J]}(y,t). \end{aligned} \tag{1.8}$$

[9] straipsnyje nagrinėjama tema yra aktuali ir pritaikoma medicinoje (analogiškas struktūras sudaro žmogaus ir kitų žinduolių kraujagyslės) ir kitose srityse.

Didelė dalis tyrimų yra atliekama ne tik analiziniu būdu nagrinėjant Navjė-Stokso lygtis, bet ir jas sprendžiant skaitiniais metodais. Tai lemia, kad Navjė-Stokso uždavinys dažniausiai aprašomas sudėtingomis diferencialinėmis lygtimis, kurių analiziniais metodais išspręsti neįmanoma. Tokiu atveju skysčių tekėjimo uždavinių sprendimui yra pasitelkiami skaitiniai metodai ir specializuota programinė įranga. [1] straipsnyje yra nagrinėjamas sudėtingo kontūro išskaidymo korektiškumas į paprastesnius kontūrus. Toks išskaidymas leidžia skysčių tekėjimo uždavinį skaitiniu būdu spręsti keliuose paprastesniuose ir mažesniuose kontūruose vietoje vieno sudėtingo, ir tada sujungti gautus rezultatus. Straipsnyje yra apibrėžiamas grafas

$$B = \cup_{j=1}^{n} e_j \tag{1.9}$$

Apibrėžiami atviri cilindrai B_j^ε su aukščiais e_j:

$$B_j^\varepsilon = \left\{ x \in \mathbb{R}^s : x_1^{e_j} \in (0, |e_j|), \left(\frac{x_2^{e_j}}{\varepsilon}, ..., \frac{x_s^{e_j}}{\varepsilon} \right) \in b_j \right\} \tag{1.10}$$

Čia, ε - cilindro skersmuo. Apibrėžiama apribota sritis:

$$\gamma_j^\varepsilon = \left\{ x \in \mathbb{R}^s : \frac{x - O_j}{\varepsilon} + O_j \in \gamma_j, j = 0, ..., n \right\} \tag{1.11}$$

Tada vamzdelių struktūra ant grafo B gali būti apibrėžiama taip:

$$B^\varepsilon = [(\cup_{j=1}^{n} B_j^\varepsilon) \cup (\cup_{j=0}^{n} \gamma_j^\varepsilon)]' \tag{1.12}$$

Tokiam plonų vamzdelių kontūrui [1] apibrėžiama teorema:

1.1.1 teiginys. *Kiekvienam sveikajam skaičiui K egzistuoja konstanta C_K, nepriklausanti nuo ε, tokia, kad su kiekvienu $\delta = C_K \varepsilon |ln\varepsilon|$ yra teisingas įvertis:*

$$||U_{\varepsilon,\delta} - u_\varepsilon||_{H^1(B^\varepsilon)} = O(\varepsilon^K) \tag{1.13}$$

Atsižvelgiant į tai, kad dalinių išvestinių diferencialines lygtis dažnai neįmanoma išspręsti analiziniu būdu, praeityje skysčių ir dujų judėjimo uždaviniai buvo sprendžiami pasitelkiant eksperimentus, o atsiradus technologijoms, leidžiančioms atlikti didelį kiekį skaičiavimų per trumpą laiką, buvo pradėta vis labiau naudoti skaitinius metodus ir kompiuteriniu būdu modeliuoti skysčių ir dujų tekėjimo uždavinius. Skaitiniu būdu skysčio tekėjimo dalinių išvestinių diferencialinės lygtys sprendžiamos iš anksto sudaryto tinklo, dengiančio dominantį kontūrą, taškuose ir tokiu būdu gaunamas apytikslis sprendinys.

Skaitinius diferencialinių lygčių sprendimo metodus nagrinėja daugybė mokslininkų matematikų ir informatikų, o skaitinį Navjė Stokso lygčių sprendimą nagrinėja Dmitri Kuzmin ([8]), Rainer Ansorge ir Thomas Sonar ([2]) ir kiti mokslininkai.

Šiame darbe yra nagrinėjamas skaitinis Navjė Stokso lygčių sprendimas panaudojant atvirojo kodo programinę įrangą OpenFoam pasirinktuose Y formos kontūruose.

OpenFOAM yra atvirojo kodo programinė įranga, susidedanti iš C++ bibliotekų, klasių ir metodų, skirtų diferencialinių lygčių aprašančių skysčių tekėjimą, šilumos sklidimą ir kitus uždavinius, sprendimui. OpenFOAM programinė įranga pradėta vystyti 2004 metais ir sulaukia vis daugiau mokslininkų ir privataus sektoriaus įmonių atstovų dėmesio, ieškančių pakankamai galingos programinės įrangos alternatyvos licencijuojamai programinei įrangai. OpenFOAM programinė įranga veikia Linux operacinėse sistemose. Kartu su OpenFOAM programine įranga platinama aibė sprendimų, skirtų pradinių duomenų ir nagrinėjamo kontūro apdorojimui (angl. pre-processing) ir gautų rezultatų atvaizdavimui ir apdorojimui (angl. post-processing).

1.2 Darbo tikslai ir uždaviniai

Magistro baigiamajam darbui buvo keliami šie tikslai:

- išanalizuoti ir aprašyti nestacionarų skysčių tekėjimą programinės įrangos OpenFOAM pagalba sprendimo principus;
- skaitiškai įvertinti slėgio šuolius kontūro išsišakojimuose, įvertinti nuo kokių veiksnių priklauso slėgio šuolio dydis.

Siekiant šių tikslų, darbo metu buvo keliami šie uždaviniai:

- susipažinti su skysčio tekėjimą aprašančiomis lygtimis, jų analiziniu ir skaitiniu sprendimu, išanalizuoti susijusią literatūrą ir mokslinius straipsnius;
- išanalizuoti OpenFOAM programinės įrangos veikimą, išmokti OpenFOAM programine įranga spręsti skirtingo tipo skysčio tekėjimo uždavinius;
- atlikti įvairius eksperimentus siekiant įvertinti slėgio šuolių skysčiui tekant Y formos kontūru priklausomybę nuo įvairių uždavinio ir kontūro parametrų: kontūro išsišakojimų nukrypimo nuo įėjimo kampas, pradinio greičio srauto įėjime dydis, skaičiavimams naudojamo tinklo stambumas.

2 skyrius

Teorinė dalis

2.1 Navjė-Stokso uždavinys

Navjė-Stokso lygtis aprašanti skysčio tekėjimą srityje $\Omega \in R^n, n = 2, 3$, kur $\partial\Omega$ yra skysčio tekėjimo srities kraštas apibrėžiama lygtimi:

$$\begin{cases} \rho \mathbf{v}_t - \nu \Delta \mathbf{v} + \rho(\mathbf{v} \cdot \nabla)\mathbf{v} + \nabla p = \rho f, (x,t) \in \Omega \times (0,T) \\ div\ \mathbf{v} = 0, (x,t) \in \Omega \times (0,T) \\ \mathbf{v}|_{S^T} = \alpha, \mathbf{v}(x,0) = \mathbf{v}_0(x) \end{cases} \tag{2.1}$$

Čia $\mathbf{v} = \mathbf{v}(x,t) = (v_1(x,t), ..., v_n(x,t))$ — skysčio greičio vektorius taške $x \in \Omega$ ir laiko momentu tm, $p = p(x,t)$ — skysčio slėgis, $\nu > 0$ — skysčio klampumo koeficientas — pastovus dydis priklausantis nuo skysčio savybių, $\rho > 0$ — skysčio tankis, $\mathbf{f} = \mathbf{f}(x,t) = (f_1(x,t), ..., f_n(x,t))$ — išorinės jėgos veikiančios skystį, $(0,T)$ — laiko intervalas kuriame nagrinėjamas skysčio tekėjimas. Šioje formulėje laikoma, kad skystis yra homogeninis (nespūdus), taigi jo tankis ρ yra pastovus dydis. 2.1 lygčių sistemoje $\mathbf{v}_t$ reiškia greičio vektoriaus $\mathbf{v}(x,t)$ išvestinę pagal laiką. Δ — Laplaso operatorius ir mūsų atveju:

$$\Delta \mathbf{v} = \frac{\partial^2 \mathbf{v}(x,t)}{\partial x_1^2} + ... + \frac{\partial^2 \mathbf{v}(x,t)}{\partial x_n^2}$$

∇ — gradientas kintamojo x atžvilgiu:

$$\nabla = \begin{pmatrix} \frac{\partial}{\partial x_1} \\ ... \\ \frac{\partial}{\partial x_n} \end{pmatrix}, \nabla p(x,t) = \begin{pmatrix} \frac{\partial p(x,t)}{\partial x_1} \\ ... \\ \frac{\partial p(x,t)}{\partial x_n} \end{pmatrix}$$

$div\mathbf{v}(x,t)$ — greičio vektoriaus divergencija, kuri apibrėžiama taip:

$$div\ \mathbf{v}(x,t) = \frac{\partial v_1(x,t)}{\partial x_1} + ... + \frac{\partial v_n(x,t)}{\partial x_n}$$

Žymėjimas „$\cdot$“ reiškia skaliarinę daugybą ir apibrėžiamas erdvėje R^n taip:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{j=1}^{n} a_j b_j$$

Salyga

$$\mathbf{v}(x,t)|_{S^T} = \alpha(x,t), kai\ x \in \partial\Omega$$

apibrėžia klampaus skysčio tekėjimą prie srities krašto $\partial\Omega$. Ši sąlyga vadinama kraštine sąlyga.

Pradinis greitis laiko momentu aprašomas pradine sąlyga:

$$\mathbf{v}(x,0)|_{S^T} = \mathbf{v}_0(x), kai\ x \in \Omega$$

Pirmoji 2.1 sistemos lygtis aprašo skysčio judėjimą, o antroji ($div\ \mathbf{v}(x,t)$) masės tvermės dėsnį. Likusios dvi lygtys apibrėžia kraštinę ir pradinę sąlygas.

2.1.1 Navjė-Stokso lygčių sprendimas skaitiniais metodais

Šiame ir tolimesniuose skyriuose siekiant suvienodinti naudojamus žymėjimus, su kituose straipsniuose naudojamais žymėjimais, greičio vektorius bus žymimas $U = (u_1, ..., u_n)$, čia $n = 2, 3$. Bendrinė transporto (skysčio tekėjimo) lygtis tokiu atveju užrašoma taip:

$$\frac{\partial U}{\partial t} - \nu\Delta U + (U \cdot \nabla)U = -\nabla p \tag{2.2}$$

Masės tvermės dėsnio lygtis užrašoma taip:

$$\nabla \cdot U = 0 \tag{2.3}$$

2.2 ir 2.3 lygtys kartu sudaro Navjė Stoksto lygtį, kuri gali būti išsprendžiama skaitiniais metodais.

Norint skaitiškai išspręsti Navjė Stokso uždavinį, pirmiausiai iš 2.2 ir 2.3 turi būti išvedama slėgio p lygtis, o tada uždavinys išsprendžiamas vienu iš pasirinktų prediktoriaus-korektoriaus algoritmų:

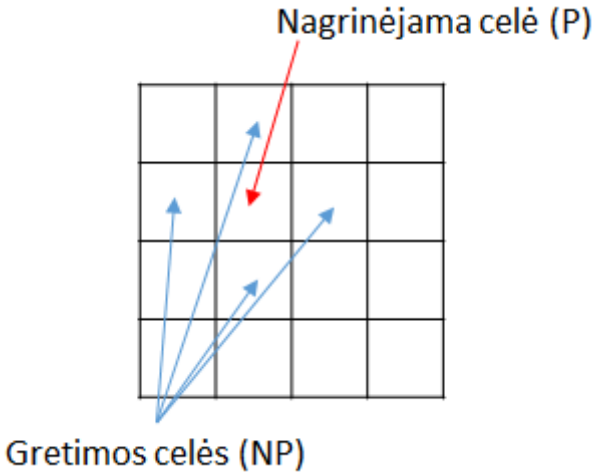
- PISO — skirtas nestacionariems skysčio tekėjimo uždaviniams. Algoritmas plačiau aprašomas 2.1.2.7 skyriuje;
- SIMPLE — skirtas stacionariems skysčio tekėjimo uždaviniams;
- PIMPLE — skirtas spręsti uždaviniams su didesniu laiko intervalu Δt apjungiantis PISO ir SIMPLE algoritmus.

Siekiant išreikšti slėgį p, visų pirma 2.2 lygtis Eulerio metodu užrašoma taip:

$$\frac{U_P^{n+1} - U_P^n}{\partial t} = -a'_P U_P^{n+1} + \sum_{NP} a'_{NP} U_{NP}^{n+1} - \nabla p \tag{2.4}$$

2.5 lygtyje P žymima konkreti tinklo celė, kuriai skaičiuojama reikšmė (žr. 2.1 paveikslą). NP žymimos kaimyninės celės, kurios yra naudojamos konkrečios celės P reikšmės skaičiavimui.

Narius prie U_P^{n+1} perkeliame į kairiąją lygties pusę, o likusius narius į dešiniąją



2.1 pav.: Celė ir kaimyninės celės

lygties pusę:

$$\left(\frac{1}{\partial t} + a'_P\right) U_P^{n+1} = \sum_{NP} a'_{NP} U_{NP}^{n+1} + \frac{1}{\partial t} U_P^n - \nabla p \tag{2.5}$$

Užrašius matricų pavidalu gauname:

$$a_P U_P = H(U) - \nabla p \tag{2.6}$$

Iš čia išreiškus U_P turime:

$$U_P = \frac{1}{a_P} H(U) - \frac{1}{a_P} \nabla p \tag{2.7}$$

Padauginus 2.7 lygtį skaliariškai iš ∇ operatoriaus ir gautą lygtį apjugus su 2.3 gaunama lygtis:

$$\nabla \cdot \left(\frac{1}{a_P} \nabla p \right) = \nabla \cdot \left(\frac{1}{a_P} H(U) \right) \tag{2.8}$$

2.8 lygtyje a_P ir $H(U)$ yra žinomi iš ankstesnės iteracijos.

2.1.2 Navjė-Stokso lygčių sprendimas OpenFOAM programine įranga

2.1.2.1 Kontūro apibrėžimas

Siekiant išspręsti skysčio tekėjimo uždavinį pasirinktame kontūre, visų pirma reikia programinių priemonių pagalba apibrėžti norimą kontūrą. *OpenFOAM* savo sudėtyje turi modulį *blockMesh* skirtą nesudėtingų kontūrų sudarymui. Jo naudojimas plačiau aprašomas [3]. *OpenFOAM* standartinis kontūro sudarymo įrankis *blockMesh* turi tik pačias paprasčiausias kontūro sudarymo galimybes ir tinka tik paprasčiausių uždavinių sprendimui.

Tuo atveju jeigu sprendžiamas uždavinys reikalauja sudėtingesnio kontūro arba naudotojui patogiau kontūrą sudarinėti grafinėje sąsajoje, reikalinga tam tikslui naudoti specializuotą *CAD* programinę įrangą.

Šio darbo apimtyje kontūras buvo apibrėžiamas *Gmsh* programinės įrangos pagalba. *Gmsh* - atviro kodo sprendimas veikiantis tiek Windows, tiek Linux operacinėse sistemose ir leidžiantis apibrėžti sudėtingesnius kontūrus, tačiau vis dar turintis tam tikrų apribojimų. Iš pagrindinių apribojimų būtų galima išskirti:

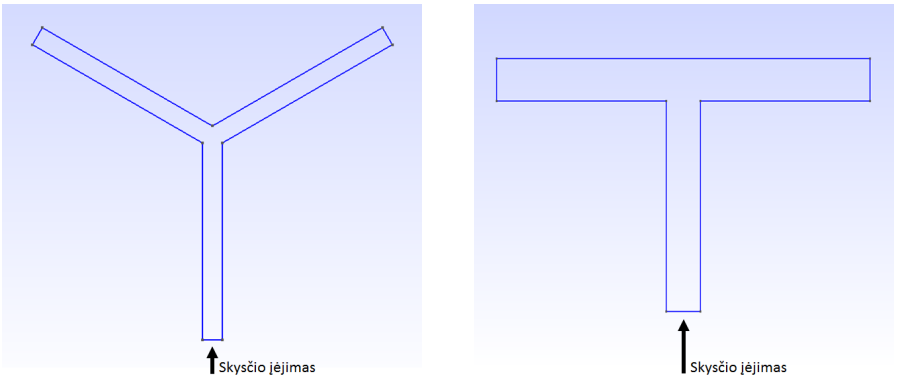
- *Gmsh* nepalaiko sudėtingų dviejų kontūrų tarpusavio geometrinių operacijų tokių kaip sąjunga, skirtumas ir atimtis. Tuo atveju, kai siekiant sudaryti sudėtingesnį kontūrą yra reikalingas paprastesnių kontūrų apjungimas, reikalinga naudoti galingesnius *CAD* sprendimu juos integruojant su *GMSH* tolimesniam apdorojimui;

- *Gmsh* naudotojo sąsajoje yra prieinama maža dalis įrankio funkcijų. Daugeliu atvejų vis dar tenka dalį kontūro apibrėžimo veiksmų atlikti naudojant teksto redaktorių;

- nepalaikomi vidiniai kintamieji, kurie būtų galimi panaudoti generuojant sudėtingesnius kontūrus;

- nepalaikoma *undo* funkcija. T.y. padarius klaidą geometrijos apibrėžimo metu, vienintelis būdas ją ištaisyti — rankiniu būdu teksto redaktoriaus pagalba ištrinti paskutinius pakeitimus *.geo rinkmenoje.

Šio darbo apimtyje buvo apibrėžtos trys Y ir viena T formos geometrija. Geometrijos pavaizduotos pavaizduotos 2.2 paveiksle. Abiejų kontūrų atveju skystis į kontūrą



2.2 pav.: Eksperimentams naudojamos geometrijos

patenka per apatinį įėjimą pastoviu greičiu $\mathbf{v}(x)$ ir toliau laisvai juda kontūru. Y kontūro atveju, kontūro šakos nuo centro yra nukrypusios per $30°, 45°$ arba $60°$. T formos kontūro atveju išėjimai su įėjimu sudaro 90 laipsnių kampą.
Kontūro apibrėžimo metu *Gmsh* programinėje įrangoje turi būti nurodomi skysčio įėjimai ir išėjimai, kontūro sienos, ribojančios skysčio tekėjimą ir geometrija sudaranti kontūro tūrį.

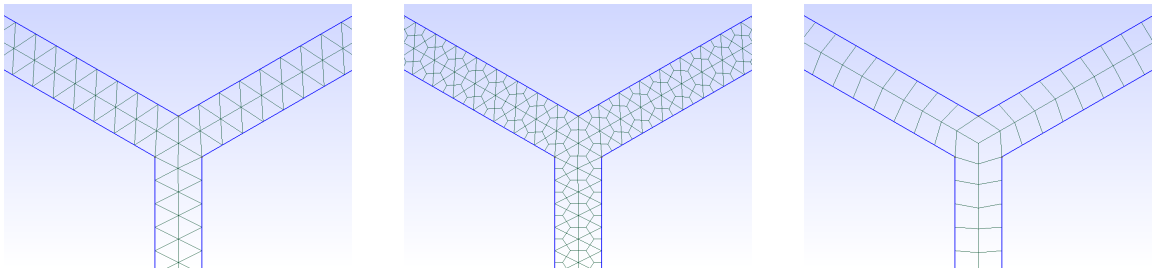
2.1.2.2 Tinklelio apibrėžimas

Turint apibrėžtą kontūrą, sekantis žingsnis yra kontūrą padengti tinklu, kurio pagalba vėliau bus ieškoma skaitinio uždavinio sprendimo. *OpenFOAM* tam tikslui yra skirtas *blockMesh* modulis, tačiau *blockMesh* naudojimas nėra patogus ir šio darbo apimtyje buvo pasirinkta tinklą generuoti naudojant pažangesnį *Gmsh* įrankį.
Toliau šiame darbe yra nagrinėjami 4 skirtingi tinklai, kurių pagalba gauti rezultatai vėliau yra palyginami tarpusavyje:

- Delauny — tinklas sudarytas iš trikampių;
- Frontal — analogiškas tinklas *Delauny* tinklui, tačiau generuojamas naudojant algoritmą lėtesnį negu Delauny atveju. Taip pat ;
- Hexes — tinklas sudarytas iš trikampių persidengiančių su šešiakampiais

- Keturkampis tinklas — tinklas sudarytas iš keturkampių elementų.

Toliau pateikiami sugeneruoti pavyzdiniai tinklai *Gmsh* nustatymuose visais atvejais pasirinkus tą patį tinklo grubumą. Paveiksle 2.3 pateikti tinklai yra tik iliustracinio



Delauny, Frontal | Hexes | Keturkampis

2.3 pav.: Skirtingi tinklai

pobūdžio ir realiems skaičiavimams buvo naudojamos šių tinklų smulkesnės versijos. Tinklų sudarymo algoritmai plačiau analizuojami 3.1 skyriuje

2.1.2.3 Tinklo tankumo nustatymas

Apibrėžiant tinklą labai svarbu atsižvelgti ir į tolimesnius uždavinio sprendimo parametrus. T.y. svarbu įvertinti, kad generuojamas tinklas atitiktų įvairias jam keliamas sąlygas ir reikalinga rinktis tarp rezultatų tikslumo (smulkinant tinklą) ir skaičiavimų greičio. Viena iš sąlygų, kuri yra svarbi skaičiavimuose yra „Courant“ skaičius. *Courant* skaičius apibrėžiamas taip:

$$C = \frac{u\Delta t}{\Delta x}$$

Čia u yra skysčio tekėjimo greitis, Δ - pasirinktas žingsnis laiko kintamojo atžvilgiu, Δx atstumas tarp dviejų artimų tinklo taškų.

2.1.1 apibrėžimas. *Courant skaičius yra įvertis to kiek judesio informacijos (u) pasikeičia vienoje tinklo elemente (Δx) per iš anksto pasirinktą laiko tarpą (Δt). Taigi, pavyzdžiui Eulerio metodų atveju Courant skaičius negali būti didesnis negu* 1*, nes tokiu atveju tai reikštų, kad vienos iteracijos metus per pasirinktą laiko tarpą Δt, skystis gali „prašokti“ mažiausią tinklo elementą ir tokiu atveju visas modelis taptų nestabilus.*

Mūsų uždavinio atveju matosi, kad generuojant tinklą reikia pasirinkti atitinkamą tinklo elemento dydį atsižvelgiant į planuojamą Δt ir/arba renkantis tinklo stambumą atsižvelgti ir į turimus skaičiavimo resursus, nes mažas Δx reikalauja atitinkamai mažo Δt.

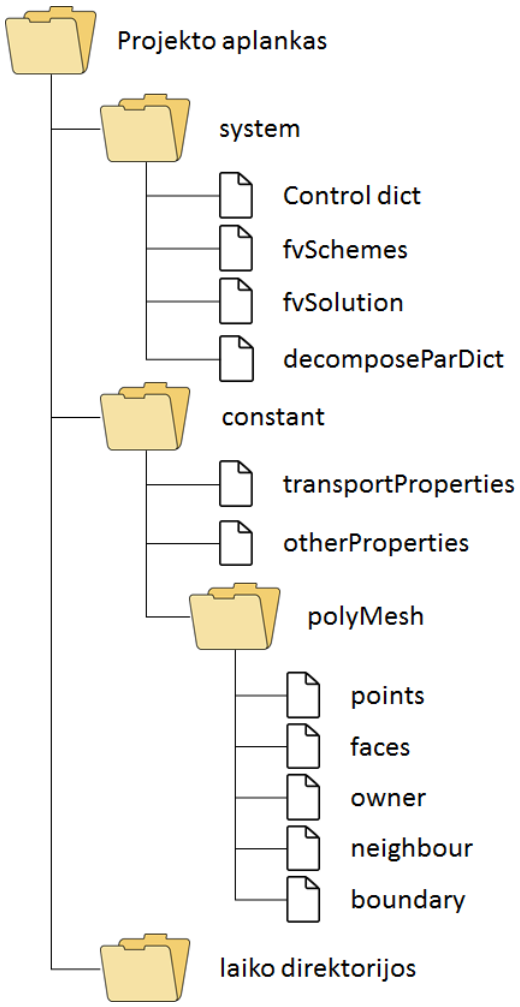
2.1.2.4 Uždavinio sprendimas OpenFOAM programine įranga

Turint parengtą tinklą, kurio pagalba bus sprendžiamas uždavinys, pereinama prie uždavinio sprendimo. Siekiant išspręsti Navjė-Stokso uždavinį OpenFOAM pagalba reikalinga įgyvendinti šiuos žingsnius:

- importuoti tinklą, ant kurio bus sprendžiamas uždavinys;
- apibrėžti pradines ir kraštines sąlygas, kitus uždavinio sprendimui aktualius parametrus;
- parinkti tinkamą uždavinio sprendimo algoritmą. Algoritmas parenkamas atsižvelgiant į kontūrą, tinklą, skysčio tipą (Niutoninis, Neniutoninis), pradines ir kraštines sąlygas.

Minimali OpenFOAM projekto struktūra nepriklausomai nuo sprendžiamo uždavinio tipo pateikiama 2.4 paveiksle. Kiekvienoje iš šių direktorijų ir failų laikomi skirtingi duomenys būtini uždavinio sprendimui:

- system — nurodomi uždavinio sprendimo parametrai, tokie kaip Δt, laiko intervalas pagal kurį yra įrašomi rezultatai ir pan.;
- constant — aplankas kuriame yra apibrėžiamas tinklas ir skysčio parametrai tokie kaip klampumas ir pan. Taip pat neniutoninio skysčio uždavinio atveju šiame aplanke nurodomi parametrai apibrėžiantys turbulenciją;
- polyMesh — aplankas, kuriame aprašomas kontūrą dengiantis tinklas. Pagrindinės aplanke esančios bylos yra: points - apibrėžia kontūro taškų koordinates, faces - apibrėžia paviršius (plokštumas), ir jas sudarančius taškus, owner, neighbour - apibrėžiami tinko elementai, boundary - byloje apibrėžiami paviršiai ribojantys kontūrą, nurodomi jų tipai;
- Laiko direktorijos — direktorijos į kurias pasirinktu dažnumu yra įrašomi uždavinio uždavinio rezultatai. Pavyzdžiui tuo atveju, kai uždavinys sprendžiamas intervale $t = [0, 60]$ su žingsniu $\Delta t = 1$, tokių aplankų uždavinio sprendimo metu bus sukurta 100 ir kiekviename jų bus fiksuojami visi uždavinio parametrai (skysčio greitis, slėgis kiekviename tinklo taške)



Minimali OpenFOAM projekto struktūra

2.4 pav.: Projekto struktura

2.1.2.5 Pradinių parametrų suvedimas

Prieš pradedant spręsti uždavinį skaitiniu būdu, būtina nurodyti uždavinio pradinę ir kraštinę sąlygas. Šios sąlygos nurodomos pradinio laiko („0“) direktorijoje, bylose „p“ — slėgį apibrėžiančios sąlygos ir „U“ - greitį apibrėžiančios sąlygos. Tiek greitis, tiek slėgis apibrėžiami ant kiekvieno kontūro paviršiaus. Keletas sąlygų apibrėžimo pavydžių:

```
konturas
{
        type            fixedValue;
        value           uniform (0 0 0); //Ant konturo sienu greitis lygus 0 m/s
}

isejimas_dsn
{
        type            zeroGradient; //Isejimo greitis neapibreziamas
}
```

Kadangi modeliuojamas Niutoninio skysčio tekėjimas vamzdžio formos kontūre, yra žinoma, kad tokiu atveju skysčio greitis vamzdelyje pasiskirsto parabolės forma.

T.y. vamzdelio viduryje greitis yra didžiausias, o prie kontūro sienų skystis „prilimpa“ ir beveik nejuda. Toks skysčio tekėjimas OpenFOAM uždavinyje aprašomas kiekvienam kontūro įėjimo taškui atskirai. Toks pradinės sąlygos aprašymas yra labai nepatogi ir ilga procedūra, todėl OpenFOAM bendruomenės nariai sukūrė papildinį *parabolicVelocity*, kuris leidžia, jį įdiegus, panaudojant specifines komandas patogiai aprašyti pradinę sąlygą:

```
iejimas
{
        type            parabolicVelocity;
        n               (0 1 0); // skyscio tekejimo kryptis (pagal koordinaciu asis)
        y               (1 0 0); // vektoriaus kryptis i kuria puse yra nukreipta konturo siena
        maxValue        1; // maksimalus greitis (vamzdelio centre)
        value           (0 0 0);
}
```

Standartinė skysčio tekėjimo pradinė sąlyga kontūro įėjime:

```
    iejimas
    {
            type            fixedValue;
            value           uniform (0 1 0); //Pradinis greitis konturo iejime 1 m/s y koordinates kryptimi
    }
```

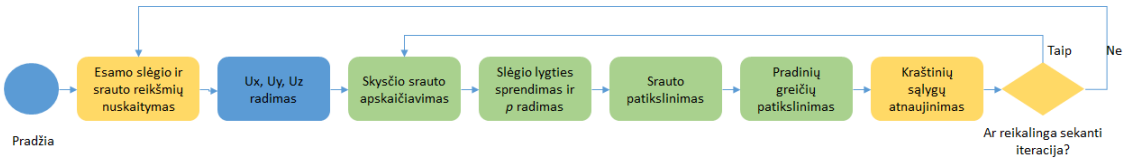
Atsižvelgiant į uždavinio specifiką, uždavinio sprendimui buvo pasirinktas *icoFoam* sprendimo variklis. *icoFoam* sprendimo variklis veikia PISO (Pressure-Implicit with Splitting of Operators) algoritmo pagrindu, kuris yra skirtas nestacionarių skysčio tekėjimo uždavinių sprendimui. Plačiau *icoFoam* sprendimo variklis ir PISO algoritmas aprašomi sekančiuose poskyriuose. Pradines ir kraštines sąlygas aprašančių bylų, uždavinio konfigūracinių bylų pavyzdžiai, pateikiami darbo prieduose.

2.1.2.6 IcoFoam sprendimo variklis

icoFoam sprendimo variklis skirtas laminaraus, nespūdaus Niutoninio skysčio tekėjimo modeliavimui. Pagrindinis algoritmas, kuriuo remiasi icoFoam atliekami skaičiavimai yra PISO, kuris detaliai aprašomas sekančiame skyriuje. Tuo atveju, jeigu modeliuojant skysčio tekėjimą yra reikalinga modeliuoti ir skysčio turbulenciją, uždavinio sprendimui vietoje *icoFoam* sprendimo variklio, gali būti panaudojamas *pisoFoam* arba *simpleFoam* varikliai.

2.1.2.7 PISO algoritmas

Raad Issa savo straipsnyje [7] apibrėžia PISO (Pressure-Implicit with Splitting of Operators) algoritmą skirtą Niutoninių skysčių nestacionaraus tekėjimo uždavinių skaitiniam sprendimui. PISO algoritmo loginė schema pateikiama 2.5 schemoje.



2.5 pav.: PISO algoritmo schema

PISO algoritmas remiasi tokiais žingsniais atliekamais kiekvienos iteracijos metu:

1. Esamo slėgio ir srauto reikšmių nuskaitymas, Courant skaičiaus apskaičiavimas — nuskaitomos paskutinės žinomos slėgio ir srauto reikšmės. U ir p reikšmės nuskaitomos panaudojant komandą:

   ```
   # include "readPISOControls.H"
   ```

   Apskaičiuojamas Courant skaičius panaudojant komandą:

   ```
   # include "CourantNo.H"
   ```

2. u_x, u_y, u_z radimas — išsprendžiama momento (transporto lygtis):

$$\frac{\partial U}{\partial t} - \nu \Delta U + (U \cdot \nabla) U = -\nabla p \tag{2.9}$$

   2.9 lygties dešinioji pusė OpenFOAM komandomis užrašoma taip:

   ```
   fvVectorMatrix UEqn
   (
     fvm::ddt(U)
     - fvm::laplacian(nu, U)
     + fvm::div(phi, U)
   );
   ```

   Čia „phi“ yra žymimas greičio vektorinio lauko projekcija į pasirinktą celės sienelę (žr. 2.1) ir apibrėžiamas taip:

   ```
   surfaceScalarField phi = linearInterpolation(U) & mesh.Sf();
   ```

   Dešinioji transporto lygties pusė yra prilyginama kairiąjai ir išsprendžiama:

   ```
   solve(UEqn == -fvc::grad(p));
   ```

   Išsprendus lygtį gaunamas apytikslis srauto įvertis visame kontūre, kuris atsižvelgiant į [4] (7.31 lygtis) apytiksliai tenkina 2.9 lygtį. Gautos U reikšmės naudojamos sekančiame žingsnyje.

3. Skysčio srauto apskaičiavimas per celės sieną su gretimomis celėmis — skaičiuojamas skysčio srautas per sieną (angl. face) iš nagrinėjamos celės į gretimą celę (žr. 2.1 paveikslą). Skysčio srautas apskaičiuojamas taip:

- Panaudojant naują U įvertį apskaičiuojamos a_P reikšmės (žr. 2.8). OpenFOAM kalboje ši operacija užrašoma taip:

```
volScalarField rUA = 1.0/UEqn.A();
```

- Apkaičiuojama $\frac{H}{a_P}$:

```
volVectorField HbyA("HbyA",U)
U = rUA*UEqn.H();
```

- Apskaičiuojamas srautas iš kiekvienos celės į kaimynines celes, per bendras celių sienas:

```
surfaceScalarField phiHbyA
(
    "phiHbyA",
    phi = (fvc::interpolate(U) & mesh.Sf()) + fvc::ddtPhiCorr(rUA, U, phi);
);
adjustPhi(phiHbyA, U, p);
```

4. Slėgio lygties sprendimas ir p radimas — sprendžiama 2.8 lygtis — $\nabla\cdot\left(\frac{1}{a_P}\nabla p\right) = \nabla\cdot\left(\frac{1}{a_P}H(U)\right)s$.

```
for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
{
    fvScalarMatrix pEqn
(
        fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
);
pEqn.setReference(pRefCell, pRefValue);
pEqn.solve();
```

5. Srauto patikslinimas — perskaičuojamas tekėjimo momentas (korektoriaus žingnsis). Žingnsio metu perskaičiuojamas ir gaunamas tikslesnis srauto per celių sienelę įvertis — $\Phi = \frac{H(U)}{a_P}\cdot S - \frac{1}{a_P}\nabla p\cdot S$.

```
if (nonOrth == nNonOrthCorr)
{
        phi = phiHbyA - pEqn.flux();
}
}
```

6. Pradinių greičių patikslinimas — tikslinama antrojo žingsnio metu gauta U reikšmė $\left(U_p = \frac{1}{a_P}H(U) - \frac{1}{a_P}\nabla p\right)$:

```
if (nonOrth == nNonOrthCorr)
        {
        phi = phiHbyA - pEqn.flux();
        }
}
```

7. Kraštinių sąlygų atnaujinimas — atnaujinamas vidinio kontūre greičio vektorius ir kraštinės sąlygos:

```
U -= rUA*fvc::grad(p);
U.correctBoundaryConditions();
```

Komanda *correctBoundaryConditions* yra naudojama, nes ankstesnėje eilutėje U reikšmės yra apskaičiuojamos pataškiu, tačiau ant kontūro krašto

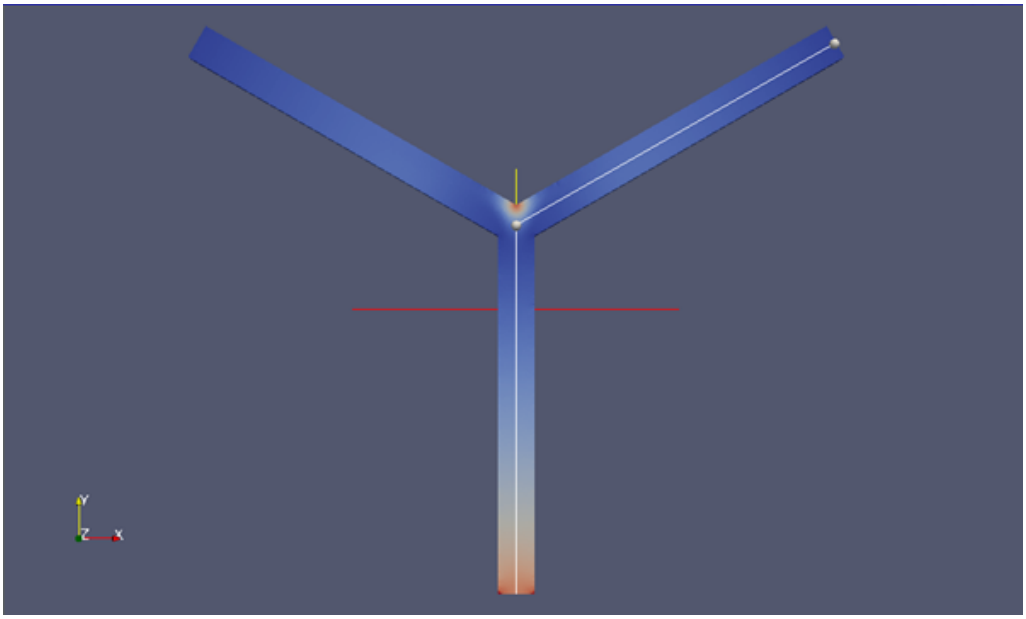
Tuo atveju, jeigu tikslumas nepakankamas, atliekama sekanti iteracija kartojant veiksmus nuo trečiojo žingsnio „Skysčio srauto apskaičiavimas“. Kitu atveju pereinama prie sekančio laiko intervalo.

2.1.2.8 Uždavinio sprendimas

Nustačius visus pradinius duomenis, OpenFOAM programinės įrangos pagalba skysčio tekėjimo uždavinys sprendžiamas paleidžiant pasirinktą sprendimo variklį. Anksčiau aprašytas icoFoam sprendimo variklis paleidžiamas Linux operacinės sistemos terminale panaudojant komandą *icoFoam*.

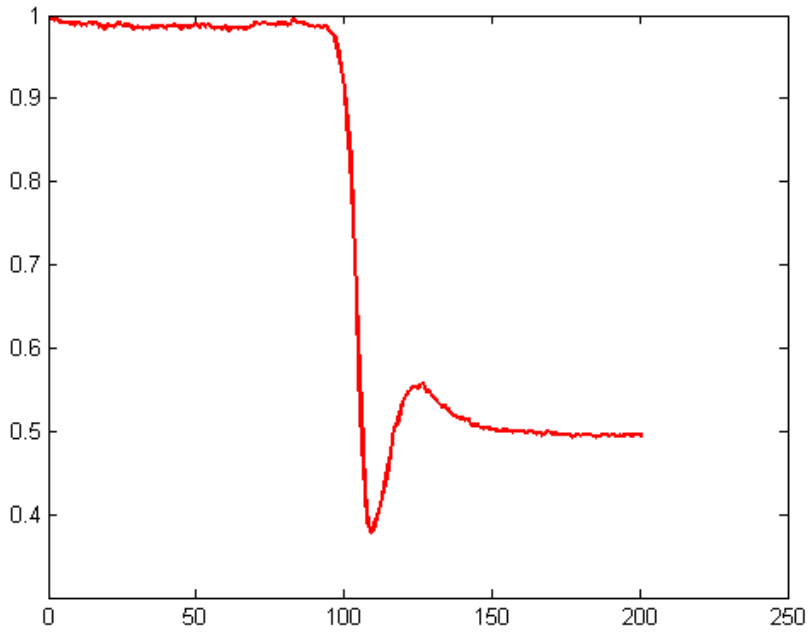
2.1.2.9 Rezultatai ir jų interpretavimas

OpenFoam programinė įranga yra platinama kartu su rezultatų interpretavimo įrankiu *paraFoam*, kuris atsakingas už vizualinį išspręsto uždavinio duomenų atvaizdavimą. *paraFoam* leidžia rezultatus analizuoti įvairiais pjūviais. Šio darbo apimtyje gauti rezultatai analizuojami ant kontūro, kuriame buvo spręstas uždavinys nubraižant grafą ir gautus rezultatus nagrinėjant ant pasirinkto grafo, žr. 2.6.

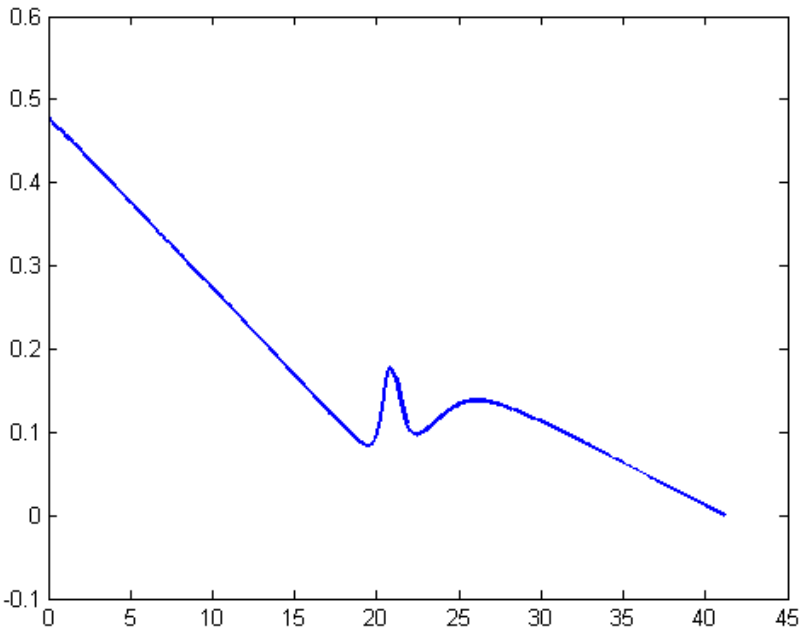


2.6 pav.: Grafas ant kontūro

Ant pasirinkto grafo gali būti nagrinėjami visi uždavinio parametrai. Mūsų atveju tai yra greitis U ir slėgis p. 60° šakų nukrypimo nuo pagrindo uždavinio atveju greičio ir slėgio kitimo grafe grafikai pavaizduoti 2.7, 2.8 paveiksluose.



2.7 pav.: Greičio grafikas ant grafo



2.8 pav.: Slėgio grafikas ant grafo

Paveiksluose matomą greičio kitimą ir slėgio šuolį kontūro išsišakojime, analiziškai plačiau nagrinėja G. Panasenko ir K. Pileckas ([9]).

3 skyrius

Mokslinis tyrimas

3.1 Skirtingų tinklų palyginimas

Atsižvelgiant į tai, kad skaitiniais metodais ieškant skysčio tekėjimo uždavinio sprendinio, skaičiavimai yra atliekami ant tinklo, kuris padengia nagrinėjamą kontūrą. Dėl šios priežasties, gauti rezultatai ir jų tikslumas labai stipriai priklauso ne tik nuo pasirinktų metodų, bet ir nuo tinklo, kuris yra naudojamas padengti nagrinėjamą kontūrą. Darbe yra nagrinėjami 4 skirtingo tipo tinklai, kurių pavyzdžiai pateikti 2.1.2.2 skyriuje. Skirtingų tinklų pagalba buvo sprendžiamas skysčio tekėjimo uždavinys Y formos kontūre, kurio išsišakojimai buvo nukrypę nuo įėjimo per 60 laipsnių. Buvo nagrinėjami šių tipų tinklai:

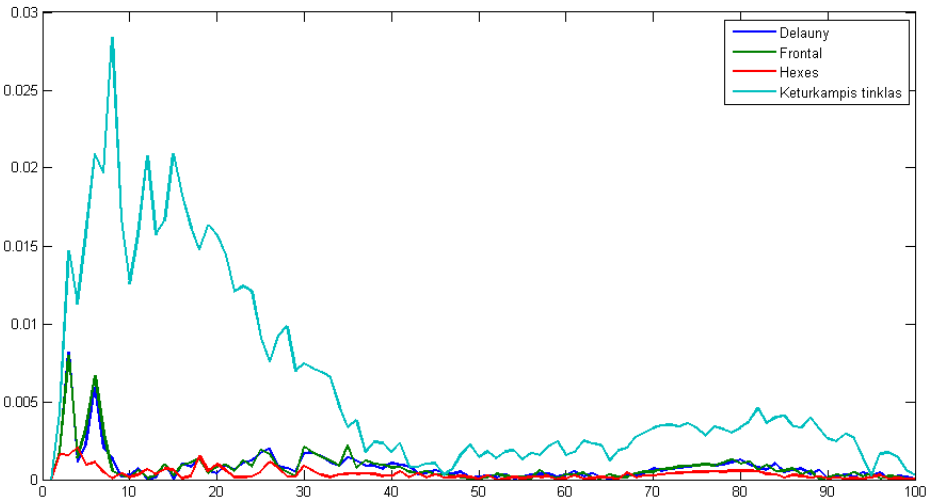
- Delauny
- Frontal
- Hexes
- Keturkampis tinklas

Išsprendus uždavinius panaudojant šiuos tinklus, kiekvienam uždaviniui buvo suformuotas grafas ir įvertinamos gautos reikšmės ant grafo. Rezultatų korektiškumas išanalizuotas dviem būdais:

- Palyginant kaip skiriasi rezultatai dešiniojoje ir kairiojoje šakose;
- Palyginant skirtingų tinklų rezultatus tarpusavyje.

3.1.0.10 Rezultatų palyginimas skirtinguose išėjimuose

Siekiant įvertinti tinklo rezultatų patikimumą, imami to konkretaus tinklo rezultatai ant kairiojo ir dešiniojo išėjimų ir tarpusavyje palyginami (atsižvelgiant į tai, kad uždavinys simetriškas, rezultatai (skysčio tekėjimo greitis ir slėgis) skirtinguose išėjimuose turi sutapti). Tuo atveju, jeigu rezultatai skiriasi stipriai daroma išvada, kad jie nėra patikimi. Jeigu rezultatai skiriasi minimaliai, daroma išvada, kad atitinkami rezultatai yra patikimesni.



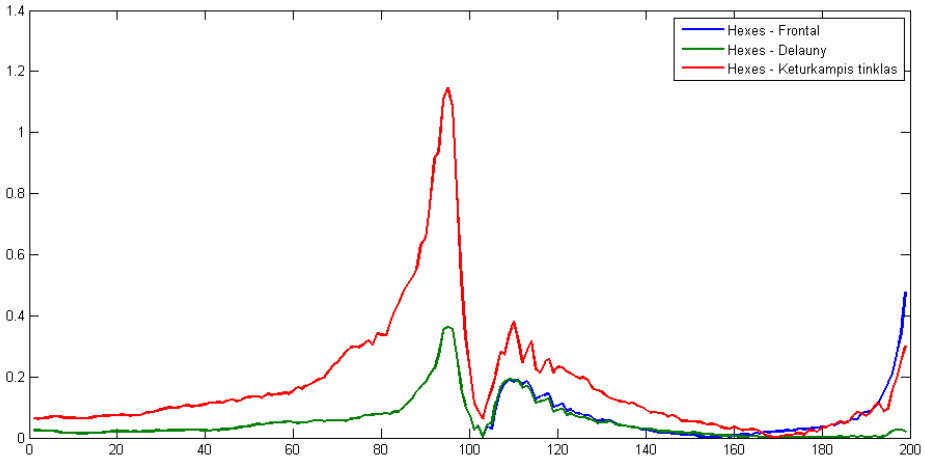
3.1 pav.: Paklaidos skirtingose grafo šakose

3.1 schemoje pateikiamas skirtumas gaunamas iš slėgio ant dešiniosios grafo šakos atėmus slėgį ant kairiosios grafo šakos pataškiui. Gautas rezultatas siekiant palyginamumo grafike braižomas atlikus modulio operaciją.
Išanalizavus pateiktą grafiką, aiškiai matosi, kad tinklas sudarytas iš keturkampių elementų yra mažiausiai tikslus, o labiausiai tikslus yra *Hexes* tinklas sudarytas iš trikampių ir šešiakampių elementų.

3.1.0.11 Skirtingų tinklų rezultatų palyginimas

Ankstesniame skyriuje buvo pateikiamas atskirų tinklų pagalba gautų rezultatų palyginimas skirtinguose tačiau simetriškuose išėjimuose. Šiame skyriuje bus lyginami skirtingais tinklais gauti rezultatai tarpusavyje. Atsižvelgiant į ankstesniame skyriuje gautą išvadą, kad *Hexes* tinklas yra labiausiai tikslus iš nagrinėtų, rezultatai gauti ant *Hexes* tinklo yra imami kaip etalonas, ir iš jų atimami kitų tinklų rezultatai. 3.2 grafike pateikiamos paklaidos lyginant skirtingų tinklų rezultatus tarpusavyje.

3.2 grafike paklaida vertinama pataškiui kaip absoliutus procentinis nuokrypis nuo etaloninio tinklo atveju gauto slėgio p dydžio tame taške.



3.2 pav.: Hexes tinklo rezultatų palyginimas su kitais tinklais

Analizuojant 3.2 grafiką galima matyti, kad skaičiavimų pradžioje Frontal ir Delauny tinklai turi vienodą paklaidą, o kvadratinio tinklo paklaida yra didžiausia. Tose grafo vietose, kuriose fiksuojamas staigus slėgio kitimas, visų tinklų paklaida stipriai išauga (Delauny ir Frontal tinklo atveju iki 40%, o kvadratinio tinklo atveju paklaida siekia netgi 120%). Grafui išsišakojus, Frontal ir Delauny paklaidos išsiskiria. Frontal tinklo paklaida artėjant prie grafo pabaigos eksponentiškai didėja, kai tuo tarpu Delauny tinklo rezultatai išlieka labai panašūs į etaloninio tinklo rezultatus.

3.1.0.12 Skirtingas tinklo elementų dydis

Skaičiavimų apimtis ir rezultatų patikimumas stipriai priklauso nuo pasirinkto tinklo grubumo (žingsnio erdvėje dydžio Δx). Siekiant skaitiškai įvertinti skirtumus tarp skirtingo grubumo tinklų, tam pačiam 45° kampo kontūrui buvo parengti 4 skiritngo grubumo tinklai. Tinklų charakteristikos pateikiamas 3.1 lentelėje.

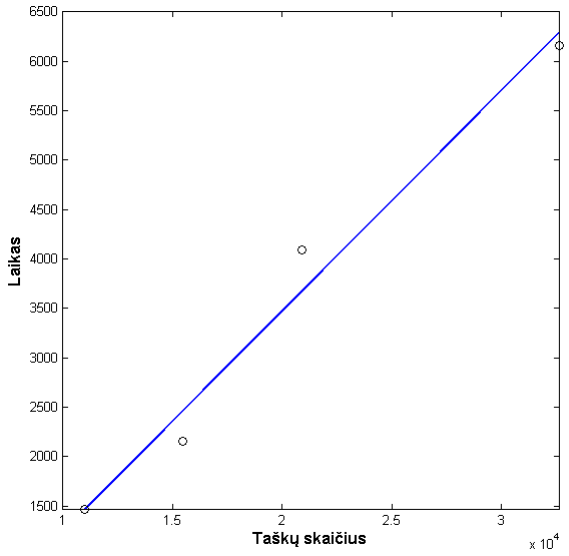
Tinklas	Taškų skaičius tinkle	Elementų skaičius tinkle
1	10998	31552
2	15466	46419
3	20898	62715
4	32624	97893

3.1 lentelė: Skirtingo grubumo tinklų charakteristikos

Atsižvelgiant į tai, kad tinklai buvo skirtingo grubumo, taip pat, siekiant išlaikyti vienodą *Courant* skaičių tarp skirtingų uždavinių, reikalinga smulkesniam tinklui rinktis smulkesnį žingsnį laike Δt. Skaičiavimų trukmė priklausomai nuo tinklo smulkumo pateikiama 3.2 lentelėje ir grafiškai atvaizduojama 3.3 grafike.

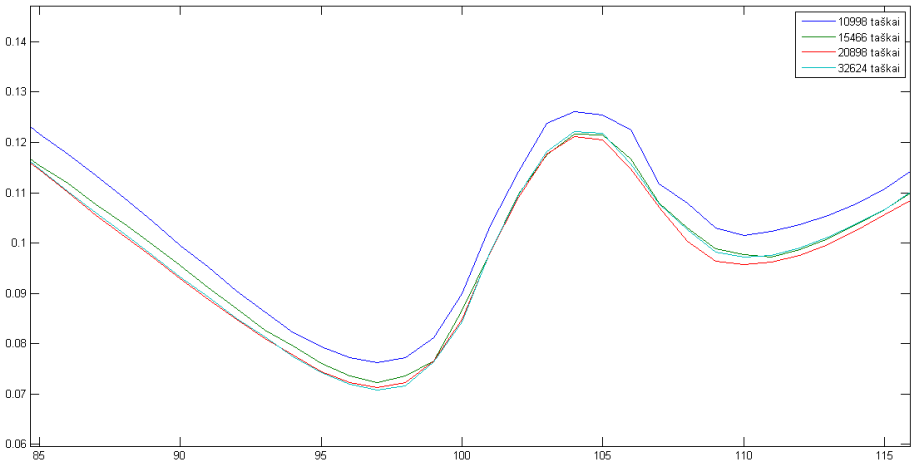
Tinklas	Taškų skaičius tinkle	Skaičiavimų laikas sekundėmis
1	10998	$1471s$
2	15466	$2153s$
3	20898	$4090s$
4	32624	$6162s$

3.2 lentelė: Uždavinio sprendimo trukmė skirtinguose tinkluose

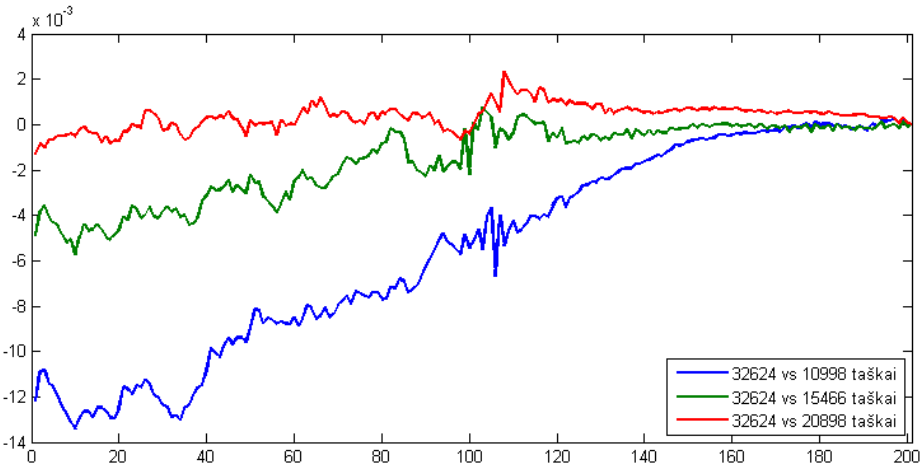


3.3 pav.: Uždavinio sprendimo trukmė priklausomai nuo tinklo tankumo

Kiekvieno skirtingo tankumo tinklo atveju gauti rezultatai pateikiami 3.4 ir 3.5 paveiksluose.



3.4 pav.: Slėgio įvertis priklausomai nuo tinklo tankumo



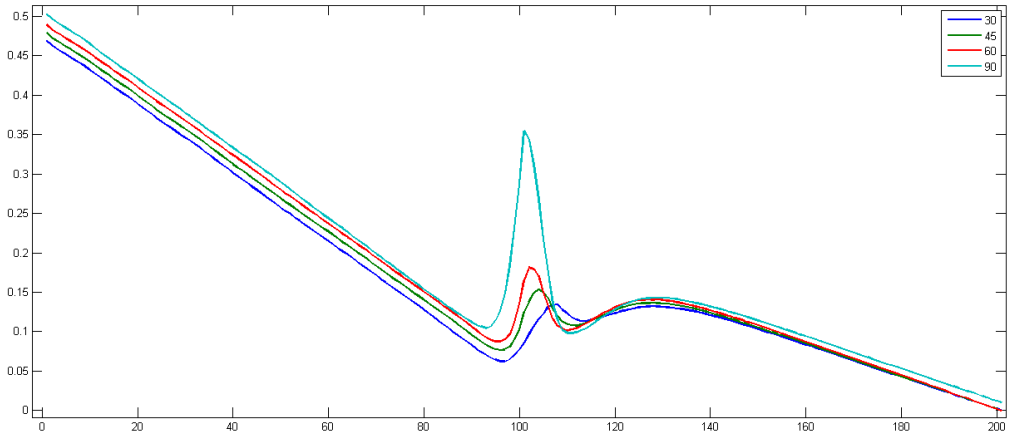
3.5 pav.: Slėgio paklaida priklausomai nuo tinklo tankumo

3.4 ir 3.5 paveiksluose yra vaizduojami gauti rezultatai rezultatai skirtingo grubumo tinkluose. Pirmame grafike yra vaizduojami slėgio rezultatai grafo išsišakojime (vaizdas yra padidintas atsižvelgiant į mažas paklaidas tarp skirtingo grubumo tinklų). Antrajame paveiksle vaizduojamas 3-jų grubiausių tinklų paklaidų palyginimas su smulkiausiu tinklu.

3.2 Kampo įtaka slėgio šuoliui

Viena iš kontūrų ypatybių, kuri buvo nagrinėjama darbo metu, buvo kampas, kuriuo nukrypsta išėjimas nuo įėjimo ir priklausomai nuo to kampo išsišakojimo atsirandantis slėgio šuolis. Uždavinys buvo nagrinėjamas vienodo tankumo tinkluose

sugeneruotuose remiantis Delauny algoritmu.



3.6 pav.: Slėgio šuolių palyginimas skirtingo kampo kontūruose

	Minimalus slėgis	Maksimalus slėgis	Skirtumas	Santykis
30°	0.0624	0.1344	0.0720	215%
45°	0.0762	0.1533	0.0771	201%
60°	0.0874	0.1812	0.0937	207%
90°	0.0979	0.3545	0.2565	362%

3.3 lentelė: Minimalus ir maksimalus lokalus slėgis priklausomai nuo kampo

Iš 3.6 grafiko ir 3.3 lentelės matosi, kad mažiausias slėgio šuolis išsišakojime fiksuojamas (slėgio maksimumas pasiekiamas toliausiai nuo išsišakojimo) mažiausio 30° kampo atveju, o didžiausias slėgio šuolis fiksuojamas didžiausio 90° kampo atveju.

3.2.1 Išsišakojimo kampo dydžio įtaka slėgiui išsišakojimo viršūnėje

Kaip ir matėme anksčiau šiame skyriuje, kontūro išsišakojimų kampas yra susijęs su slėgio šuolio dydžiu. Ankstesniame skyriuje buvo nagrinėjamas slėgio šuolis ant grafo, kuris eina per kontūro vidurį, tačiau nebuvo nagrinėjama kaip keičiasi slėgis kontūro išsišakojimo kampo viršūnėje. 3.4 lentelėje pateikiamas slėgio įvertinimas priklausomai nuo kampo, kuriuo yra nukrypusios viršūnės.

Kontūras	Slėgis išsišakojimo viršūnėje
$30°$	$0,4$
$45°$	$0,41$
$60°$	$0,48$
$90°$	$0,55$

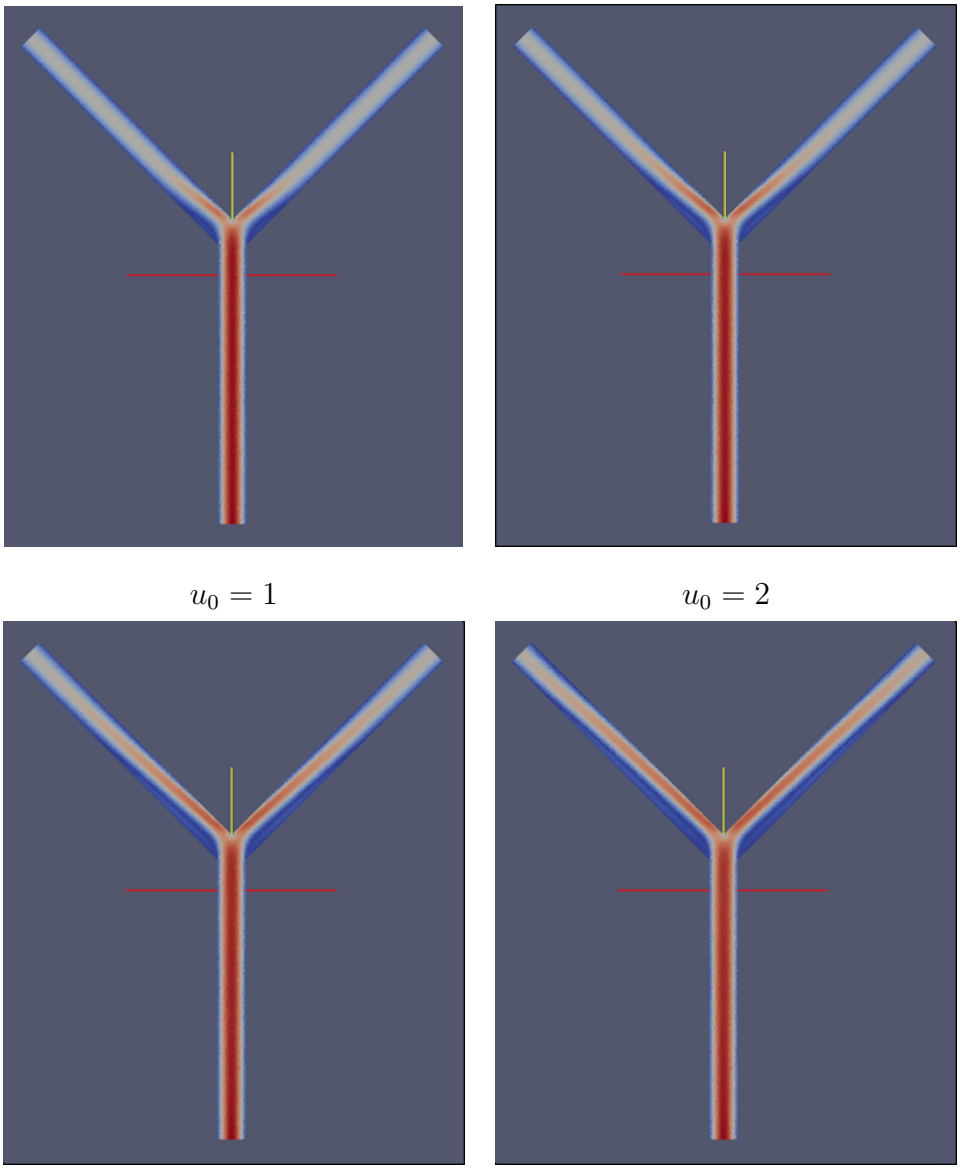
3.4 lentelė: Slėgis išsišakojimo viršūnėje priklausomai nuo kontūro kampo

Išnagrinėjus 3.4 lentelę, galima matyti, kad didėjant kontūro kampui taip pat didėja ir slėgis kampo viršūnėje, tačiau visų kontūrų atveju viršūnėje pasiekiamas maksimalus slėgis yra apytiksliai panašus. T.y. slėgio šuolio absoliutus dydis $30°, 45°, 60°$ ir $90°$ kampų viršūnėse yra gana panašus, kai tuo tarpu nubrėžus grafą einantį per kontūro centrą skiriasi daug stipriau.

3.3 Pradinio greičio įtaka slėgiui

Darbe vertinama pradinio greičio įtaka skaitiniu būdu gautiems greičio ir slėgio įverčiams. Siekiant įvertinti pradinio greičio įtaką uždavinio rezultatams buvo pasirinkta tyrinėti kontūrą su išsišakojimais nukrypusiais $45°$ laipsnių kampu ir greičiais $u = \{1; 2; 5; 10; 20\}$ Taip pat uždavinio apibrėžimo OpenFOAM metu atsižvelgta į 2.1.1 apibrėžime apibrėžtą Courant skaičiaus formuluotę, kuri reiškia, kad didėjant uždavinyje nagrinėjamiems greičiams, atitinkamai turi mažėti skaičiavimuose naudojamas žingsnis erdvėje Δx. Tai reiškia, kad didesnių pradinių greičių nagrinėjimas kontūre reikalauja didesnių skaičiavimo resursų sąnaudų.

Gauti greičio kontūrai pateikti 3.7 paveiksle. Pateiktose schemose matosi, kad esant didesniam pradiniam greičiui, išėjimuose greitis yra nukrypęs nuo centro ir nesudaro taisyklingos parabolės. Didėjant greičiui, parabolė tampa taisyklinga vis toliau nuo išsišakojimo.



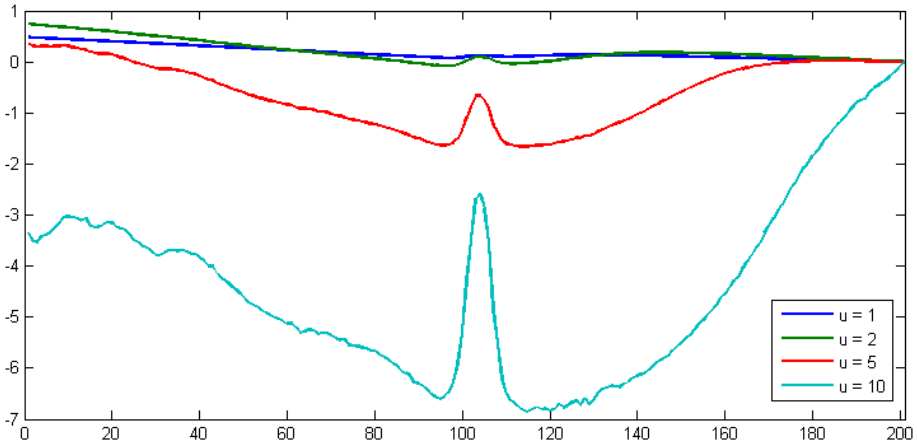
$u_0 = 1$ $u_0 = 2$

$u_0 = 5$ $u_0 = 10$

3.7 pav.: Greitis priklausomai nuo pradinio greičio

3.3.1 Slėgio ant grafo ir bifurkacijos kampo viršūnėje palyginimas

Šiame poskyryje įvertinamas slėgis ant grafo ir analizuojama jo priklausomybė nuo pradinio greičio. Schemoje 3.8 pateikiami slėgio įverčiai priklausomai nuo pradinio greičio. Iš grafiko aiškiai matosi, kad prie didesnio pradinio greičio, santykinis slėgio šuolis yra didesnis lyginant su slėgio šuoliu tuo atveju, kai pradinis greitis mažesnis.



3.8 pav.: Slėgis ant grafo priklausomai nuo pradinio greičio

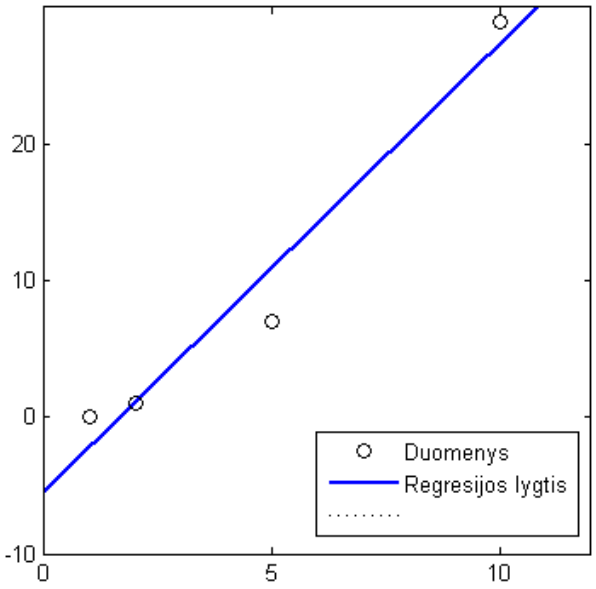
Verta atkreipti dėmesį į tai, kad 3.8 schemoje visi grafikai sueina į vieną tašką. Taip yra dėl to, kad šiame taške (išėjimų galuose) yra užduodama slėgio sąlyga (slėgis išėjime lygus nuliui).

Slėgis išsišakojimo viršūnėje

Be slėgio įvertinimo ant grafo einančio tiksliai per kontūro iėjimo ir išėjimo dalių vidurį, slėgį taip pat galima įvertinti ir kontūro išsišakojimo viršūnėje. Lentelėje 3.5 pateikiamas slėgio įvertis priklausomai nuo pradinio greičio. Paveiksle 3.9 pateikiamas grafinis pradinio ir greičio duomenų atvaizdavimas ir regresijos tiesė šiems duomenims.

Pradinis greitis	Slėgis išsišakojimo viršūnėje
$u = 1$	0.479
$u = 2$	1,48
$u = 5$	7,65
$u = 10$	29,1

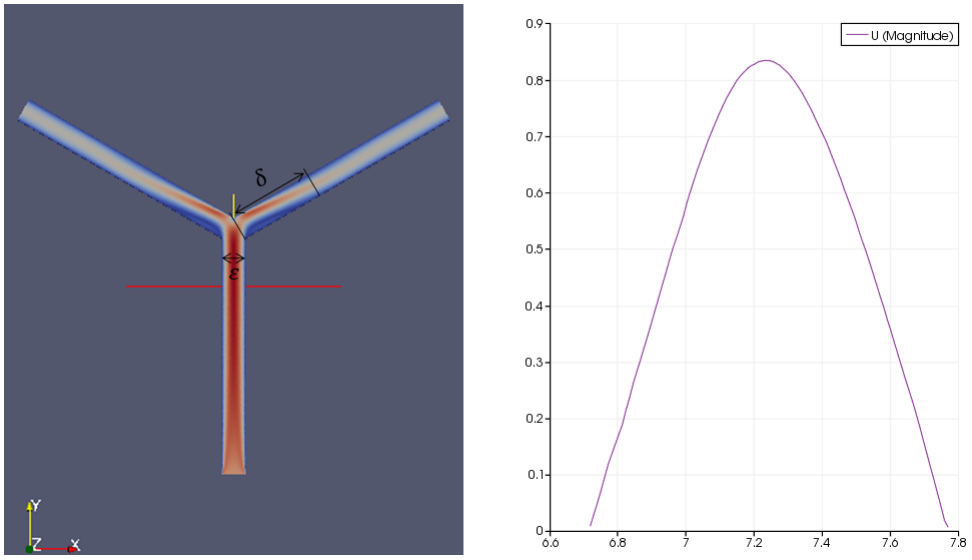
3.5 lentelė: Slėgis išsišakojimo viršūnėje priklausomai nuo pradinio greičio dydžio



3.9 pav.: Slėgis viršūnėje priklausomai nuo pradinio greičio

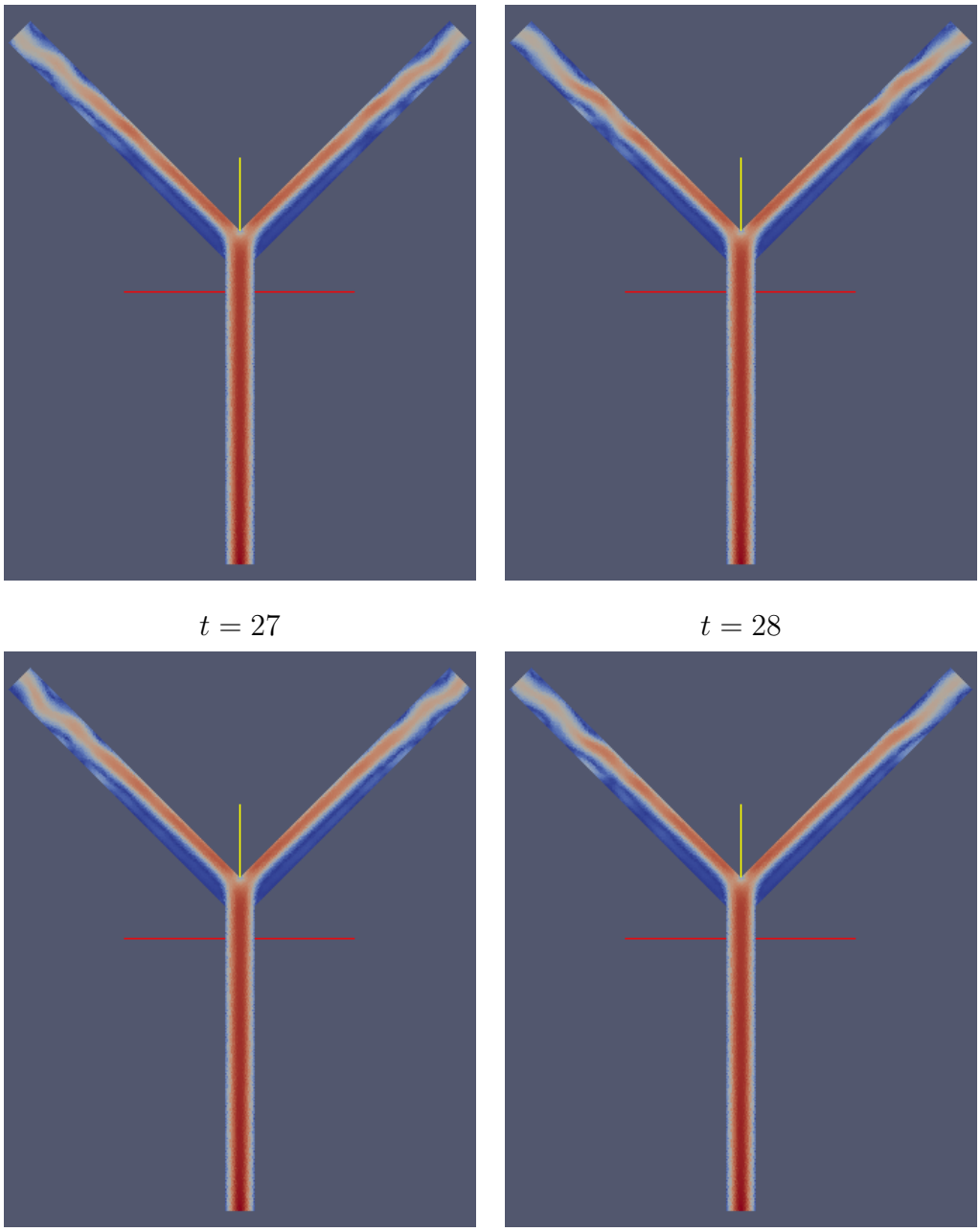
3.4 Kontūro asimptotinis išskaidymas (MAPDD)

[9], [1] straipsniuose nagrinėjama sudėtingo kontūro išskaidymo galimybė (angl. MAPDD — The method of asymptotic partial domain decomposition for thin tube structures). Paveiksle 3.10 pateikiamas grafinis MAPDD paaiškinimas. Kairėje paveikslo pusėje pateikiamas nagrinėjamas kontūras, o dešinėje pateikiamas greičio grafikas atstumu δ nuo bifurkacijos kampo. [1] nurodoma, kad δ įvertis priklauso nuo plono vamzdelio skersmens ε ir yra $\varepsilon|ln\,\varepsilon|$ eilės.



3.10 pav.: Greitis ir greičio pasiskirstymas atstumu δ nuo bifurkacijos kampo

Darbo metu buvo atlikti eksperimentai su 45° bifurkacijos kampo kontūru ir skirtingais pradiniais greičiais V_0 siekiant įvertinti ar δ priklauso nuo pradinio greičio. 3.7 paveiksle aiškiai matosi, kad didėjant pradiniam greičiui, skysčio tekėjimo greitis nusistovi ir tampa taisyklinga parabole einančia per vamzdelio centrą vis toliau nuo bifurkacijos kampo. T.y. δ yra proporcingas pradiniam greičiui v_0. 3.7 paveiksle pateikiami greičių grafikai vamzdeliuose iki maksimalaus greičio $u_0 = 10$. Pradinį greitį padidinus dvidešimt kartų iki $u_0 = 20$ gaunama situacija, kai nagrinėjamame kontūre skysčio tekėjimas per nagrinėjamą laiko tarpą nenusistovi.



$t = 27$ $t = 28$

$t = 29$ $t = 30$

3.11 pav.: Greitis kontūre, kai $u_0 = 20$

3.5 Išvados ir rezultatai

Darbo metu buvo pasiekti šie rezultatai, kurie aprašomi atsižvelgiant į 1.2 skyriuje pateiktus darbo tikslus:

- išanalizuoti ir aprašyti nestacionaraus skysčio tekėjimo programinės įrangos OpenFOAM pagalba sprendimo principai - OpenFOAM nemokama programinė įranga skirta diferencialinių lygčių sprendimui, o jos teikiamas funkcionalumas yra pakankamas tiek naudojimui universitete, tiek verslo įmonėse, tačiau OpenFOAM komercinėms programinės įrangos alternatyvoms nusileidžia savo naudotojo sąsaja ir naudojimosi patogumu (atskiriems uždavinio sprendimo žingsniams reikalingos atskiros paprogramės);

- skaitiškai įvertinti slėgio šuoliai kontūro išsišakojimuose (bifurkacijos taškuose), slėgio šuoliai ant grafo einančio per vamzdelių centrą, veiksniai, nuo kurių priklauso slėgio šuolių dydis:

  - kontūro išsišakojime fiksuojamas slėgio šuolis, kuris yra maksimalus kampo viršūnėje (bifurkacijos taške);
  - slėgio šuolio dydis fiksuojamas priklausomai nuo kampo dydžio - didesnio kampo atveju slėgio šuolis yra didesnis. „T“ formos kontūro atveju slėgis yra didžiausias iš visų tirtų kontūrų;
  - didinant bifurkacijos kampo dydį, slėgio šuolio dydžio absoliuti maksimali reikšmė bifurkacijos kampe didėja nežymiai, tačiau didesnio kampo atveju fiksuojamas didesnis santykinis slėgio šuolis ant grafo išsišakojimo. Tai leidžia daryti prielaidą, kad mažo kampo atveju slėgio šuolis yra sukoncentruotas bifurkacijos kampo viršūnėje, o didesnio kampo atveju slėgis padidėja nežymiai, tačiau apima vis didesnį plotą nagrinėjamame kontūre.

Literatūra

[1] Grigory Panasenko Abdessalem Nachit and Abdelmalek Zine. Asymptotic partial domain decomposition in thin tube structures: Numerical experiments. *Journal for Multiscale Computational Engineering*, 11(5):407–441, 2013.

[2] Rainer Ansorge and Thomas Sonar. *Mathematical Models of Fluid Dynamics*. Wiley-VCH.

[3] CFD Direct. OpenFOAM User Guide: 5.3 Mesh generation with blockMesh. `http://cfd.direct/openfoam/user-guide/blockMesh/#x25-1420005.3`. [Internetinis šaltinis; tikrinta 2016-05-03].

[4] J.H. Ferziger and M. Peric. *Computational Methods for Fluid Dynamics*. Springer Berlin Heidelberg, 2012.

[5] C. Geuzaine and Gmsh J.-F. Remacle. A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities.

[6] Francis X. Giraldo. Time-integrators.

[7] Raad Issa. Solution of the implicitly discretised fluid flow equations by operator-splitting. *Journal of Computational Physics*, 62:40–65, 1985.

[8] Dmitri Kuzmin. *A Guide to Numerical Methods for Transport Equations*.

[9] Grigory Panasenko and Konstantin Pileckas. Asymptotic analysis of the non-steady navier-stokes equations in a tube structure. 1. the case without boundary-layer-in-time.

[10] Konstantinas Pileckas. *Navjė-Stokso lygčių matematinė teorija*. Matematikos ir informatikos institutas, 2007.

A Priedas

Elektroninė laikmena

Kartu su darbu pateikiama elektroninė laikmena, kurioje yra įrašyti visi darbo rengimo metu naudoti kontūrai, uždavinius apibrėžiantys aplankai ir gauti skaičiavimų rezultatai.

B Priedas

30° kontūro apibrėžimas

```
Point(1) = {-1, 0, 0, 1.0};
Point(2) = {1, 0, 0, 1.0};
Point(3) = {1, -20, 0, 1.0};
Point(4) = {-1, -20, 0, 1.0};
Line(1) = {1, 2};
Line(2) = {2, 3};
Line(3) = {3, 4};
Line(4) = {4, 1};
Line Loop(5) = {1, 2, 3, 4};
Plane Surface(6) = {5};
Translate {0, 20, 0} {
Duplicata { Surface{6}; }
}
Delete {
Surface{6};
}
Delete {
Line{4};
}
Delete {
Line{2};
}
Rotate {{0, 0, 1}, {1, 0, 0}, -Pi/6} {
Surface{7};
}
Rotate {{0, 0, 1}, {1, 0, 0}, Pi/6} {
Duplicata { Surface{7}; }
}
Delete {
Surface{7};
}
Delete {
Line{9};
}
Delete {
Line{1};
}
Delete {
Line{11};
}
Rotate {{0, 0, 1}, {-1, 0, 0}, Pi/6} {
Surface{12};
}
Point(17) = {1, 0, 0, 1.0};
Delete {
Surface{12};
```

```
}
Delete {
Line{14};
}
Delete {
Line{15};
}
Delete {
Point{2};
}
Delete {
Point{1};
}
Line(17) = {17, 3};
Line(18) = {16, 4};
Line(19) = {17, 6};
Point(18) = {0, 1, 0, 1.0};
Rotate {{0, 0, 1}, {1, 0, 0}, Pi/6} {
Duplicata { Point{16}; }
}
Rotate {{0, 0, 1}, {1, 0, 0}, -Pi/6} {
Duplicata { Point{16}; }
}
Point(21) = {0, 2.267948973067089, 0, 1.0};
Delete {
Point{19};
}
Delete {
Point{20};
}
Delete {
Point{18};
}
Line(20) = {8, 21};
Line(21) = {21, 5};
Line Loop(22) = {20, 21, 8, -19, 17, 3, -18, 16, 13};
Plane Surface(23) = {22};
Extrude {0, 0, -1} {
Surface{23};
Layers{1};
Recombine;
}
Physical Surface("iejimas") = {57};
Physical Surface("isejimas_dsn") = {45};
Physical Surface("isejimas_kr") = {69};
Physical Surface("priekis") = {23};
Physical Surface("galas") = {70};
Physical Surface("konturas") = {61, 65, 37, 41, 49, 53};
Physical Volume("vidus") = {1};
```

C Priedas

Greičio pradinės ir kraštinių sąlygų apibrėžimas

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
        version     2.0;
        format      ascii;
        class       volVectorField;
        object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
        konturas
        {
                type            fixedValue;
                value           uniform (0 0 0);
        }
        iejimas
        {
                type            parabolicVelocity;
                n               (0 1 0);
                y               (1 0 0);
                maxValue        1;
                value           (0 0 0);
        }
        isejimas_kr
        {
                type            zeroGradient;
        }
        isejimas_dsn
        {
                type            zeroGradient;
        }
        priekis
        {
```

```
        type            empty;
    }
    galas
    {
        type            empty;
    }
}
// ************************************************************************* //
```

D Priedas

Slėgio pradinės ir kraštinių sąlygų apibrėžimas

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
version     2.0;
format      ascii;
class       volScalarField;
object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 2 -2 0 0 0 0];
internalField   uniform 0;
boundaryField
{
konturas
{
type            zeroGradient;
}
iejimas
{
type            zeroGradient;
}
isejimas_kr
{
type            fixedValue;
value           uniform 0;
}
isejimas_dsn
{
type            fixedValue;
value           uniform 0;
}
priekis
{
type            empty;
}
galas
```

```
{
type            empty;
}
}
// ************************************************************************* //
```

E Priedas

Uždavinio parametrų nustatymas

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
version     2.0;
format      ascii;
class       dictionary;
location    "system";
object      controlDict;
}
libs ("libmyBCs.so");
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application     icoFoam;
startFrom       latestTime;
startTime       0;
stopAt          endTime;
endTime         50;
deltaT          0.01;
writeControl    timeStep;
writeInterval   50;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
// ************************************************************************* //
```

F Priedas

icoFoam sprendimo variklio algoritmas

```
/*---------------------------------------------------------------------------*\
=========                 |
\\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
\\    /   O peration     |
\\  /    A nd           | Copyright (C) 2011-2013 OpenFOAM Foundation
\\/     M anipulation  |
-------------------------------------------------------------------------------

Application
icoFoam

Description
Transient solver for incompressible, laminar flow of Newtonian fluids.

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
#include "setRootCase.H"

#include "createTime.H"
#include "createMesh.H"
#include "createFields.H"
#include "initContinuityErrs.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Info<< "\nStarting time loop\n" << endl;

while (runTime.loop())
{
Info<< "Time = " << runTime.timeName() << nl << endl;

#include "readPISOControls.H"
#include "CourantNo.H"

fvVectorMatrix UEqn
(
```

```
fvm::ddt(U)
+ fvm::div(phi, U)
- fvm::laplacian(nu, U)
);

solve(UEqn == -fvc::grad(p));

// --- PISO loop

for (int corr=0; corr<nCorr; corr++)
{
volScalarField rAU(1.0/UEqn.A());

volVectorField HbyA("HbyA", U);
HbyA = rAU*UEqn.H();
surfaceScalarField phiHbyA
(
"phiHbyA",
(fvc::interpolate(HbyA) & mesh.Sf())
+ fvc::interpolate(rAU)*fvc::ddtCorr(U, phi)
);

adjustPhi(phiHbyA, U, p);

for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
{
fvScalarMatrix pEqn
(
fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
);

pEqn.setReference(pRefCell, pRefValue);
pEqn.solve();

if (nonOrth == nNonOrthCorr)
{
phi = phiHbyA - pEqn.flux();
}
}

#include "continuityErrs.H"

U = HbyA - rAU*fvc::grad(p);
U.correctBoundaryConditions();
}

runTime.write();

Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
<< "  ClockTime = " << runTime.elapsedClockTime() << " s"
<< nl << endl;
}

Info<< "End\n" << endl;

return 0;
}


// ************************************************************************* //
```