ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta strojní, Ústav technické matematiky

BAKALÁŘSKÁ PRÁCE

Numerická simulace obtékání profilu

Autor: Martin Helcl

Vedoucí práce: Doc.Ing. Jiří Fürst, PhD.

Datum: červen 2017

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Helcl** Jméno: **Martin** Osobní číslo: **439072**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav technické matematiky**

Studijní program: **Teoretický základ strojního inženýrství**

Studijní obor: **bez oboru**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Numerická simulace obtékání profilu

Název bakalářské práce anglicky:

Numerical simulation of flows over a profile

Pokyny pro vypracování:

Student se seznámí se základními rovnicemi popisujícími proudění tekutin a s principy jejich numerického řešení. Pomocí softwarového balíku OpenFOAM potom provede numerickou simulaci obtékání izolovaného profilu nestlačitelnou tekutinou a získané výsledky porovná s dostupnými experimentálními daty.

Seznam doporučené literatury:

H. Versteeg; W. Malalasekera: An Introduction to Computational Fluid Dynamics: The Finite Volume Method
M. Virius: Základy programování v C++

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Jiří Fürst Ph.D., ústav technické matematiky FS

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **13.04.2017** Termín odevzdání bakalářské práce: **18.08.2017**

Platnost zadání bakalářské práce: _____

Podpis vedoucí(ho) práce	Podpis vedoucí(ho) ústavu/katedry	Podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání	Podpis studenta
20.4.2017	

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Anotační list

Jméno autora:	Martin Helcl
Název BP:	Numerická simulace obtékání profilu
Anglický název:	Numerical simulaton of flows over a profile
Rok:	2017
Obor studia:	Teoretický základ strojního inženýrství
Ústav:	Ústav technické matematiky
Vedoucí	Doc. Ing. Jiří Fürst, PhD.
Bibliografické údaje:	počet stran: 53
	počet obrázků: 51
	počet tabulek: 0
	počet příloh: 1
Klíčová slova:	OpenFOAM, Gmsh, numerická simulace, NACA2412, metoda konečných objemů, Spalart-Allmaras, koeficient vztlaku, koeficient odporu
Key words:	OpenFOAM, Gmsh, numerical simulation, NACA2412, finite volume method, Spalart-Allmaras, drag coefficient, lift coefficient

Anotace:
Tato práce je věnována přípravě, provedení a vyhodnocení numerické simulace obtékání profilu křídla NACA 2412 při různých úhlech náběhu. Ta je provedena pomocí softwarového balíku OpenFOAM. Pozornost je hlavně věnována procesu tvorby sítě v programu Gmsh a tvorbě programu v C++ na automatické generování sítě pro daný úhel náběhu a čtyřmístný NACA profil.
Abstract:
This bachelor thesis describes the procedure of a numerical simulation of flow over a profile of an airfoil NACA2412 at different angles of attack. For that purpose open source software package OpenFOAM is used. The main focus is on creation process of a grid for the simulation using program Gmsh and creation of a program in C++ to automate creating grid for any angle of attack and four digit NACA profile.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně pod vedením Doc. Ing. Jiřího Fürsta, PhD. a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

Poděkování

Vedoucímu práce, panu Doc. Ing. Jiřímu Fürstovi, PhD., děkuji za cenné rady a podnětné připomínky. Své rodině děkuji za podporu během studia.

Obsah

1 ÚVOD

Proudění tekutin bylo odjakživa problémem, který inženýři museli řešit v mnohých aplikacích. Avšak řešení složitějších případů, hlavně co se týče komplexnosti geometrie, je velmi těžké. Rovnice popisující tento problém jsou známy již od 19. století, kdy je objevili Claude-Louis Navier a George Gabriel Stokes, po nichž byly také pojmenovány. Analytické řešení těchto nelineárních parciálních diferenciálních rovnic jsme však schopni naleznout jen pro jednoduché případy. Možným řešením je realizace experimentu, avšak tento způsob je finančně i časově velmi náročný.

Numerická řešení daných rovnic pomocí výpočetní techniky znamenala značný průlom v této problematice. Počátky používání počítačových simulací proudění tekutin se datují do 60. let 19. století. Prvním odvětvím bylo letectví, kontrétně aerodynamika letadla a konstrukce proudových motorů. Používání těchto simulací se postupně rozšířilo i do jiných technických odvětví díky vývoji výpočetní techniky, která se stala cenově dostupnější. V dnešní době jsou softwarové simulace neodmyslitelnou součástí konstrukce mnoha výrobků, ať se jedná o aerodynamické výpočty u automobilů, modelování vstřikování u spalovacích motorů, zatékání taveniny do formy a nespočet dalších aplikací [7]. Ačkoliv tato metoda pravděpodobně nikdy zcela nenahradí experiment, výrazně pomůže zlevnit i zrychlit realizaci projektu.

I s tímto řešením však přichází mnoho problémů, které musíme překonat, chceme-li docílit dostatečně přesných výsledků. Zejména při turbulentním proudění, které se bohužel vyskytuje v mnoha inženýrských aplikacích. Volba správných zjednodušujících předpokladů a modelů je klíčovým, avšak složitým úkolem, který stále dává zabrat inženýrům i vědcům.

Předmětem této práce je seznámit se se základními principy těchto výpočtů a následně jejich aplikace při simulaci obtékání profilu křídla.

2 MATEMATICKÝ POPIS PROUDĚNÍ TEKUTIN

2.1 Úvod

Úkolem této kapitoly je stručně se seznámit se základními rovnicemi popisujícími proudění tekutin a principy jejich numerického řešení. Pro tento matematický popis budeme používat Eulerova přístupu, ve kterém zkoumáme průběhy fyzikálních veličin v nepohyblivém kontrolním bodě, který je daný materiálovým souřadným systémem. Ten je pro problematiku mechaniky tekutin nejvhodnější. Dále budou všechny rovnice uváděny ve vektorovém tvaru pro kartézské souřadnice x, y, z.

2.2 Navierova-Stokesova rovnice

Navierova-Stokesova rovnice je základem matematického popisu proudění tekutin. Spolu s rovnicí kontinuity tvoří obecný popis jakéhokoliv proudění newtonské látky.

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \frac{1}{3}\nu \nabla(\nabla \cdot \mathbf{u}) + \mathbf{g} \tag{1}$$

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{2}$$

Zde $\mathbf{u}(\mathbf{x}, t)$ je vektor rychlosti proudění, $p(\mathbf{x}, t)$ tlak, $\rho(\mathbf{x}, t)$ hustota, $\nu(T)$ kinematická viskozita a $\mathbf{g}(\mathbf{x}, t)$ vektor vnějších objemových sil. Pro náš případ obtékání profilu křídla můžeme považovat vzduch za nestlačitelnou látku, ρ a ν budou tedy konstantní a zanedbáme vnější objemové síly ($\mathbf{g} = 0$). Pak rovnice (1) a (2) můžeme upravit do následujícího tvaru.

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} \tag{3}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{4}$$

Problémem je řešení této soustavy rovnic, jelikož (3) je nelineární parciální diferenciální rovnice. Exaktní řešení umíme nalézt zpravidla jen pro proudění s jedinou nenulovou složkou,

jelikož v tomto případě vymizí nelineární konvektivní člen $\mathbf{u} \cdot \nabla \mathbf{u}$ [5].

Další možnost řešení je inspekční analýza. V této metodě zavádíme bezrozměrná kritéria, která dáme do souvislostí úpravou rovnic popisujících daný problém. Konkrétní závislosti pak určujeme empiricky na základě experimentu pro konkrétní geometrii a rozsah bezrozměrných kritérií. Nutnost experimentálních dat však značně omezuje použití této metody.

Pro případy kdy nemáme tato data k dispozici, musíme použít numerická řešení, ta však přináší mnoho problémů týkajících se tvorby sítě a volby modelů turbulence.

2.3 Model turbulence

2.3.1 Úvod

Turbulence je definována jako chaotická a náhodná změna tlaku a rychlosti v prostoru a čase. Její modelování je bezpochyby největším problémem mechaniky tekutin. Přesné zachycení všech těchto turbulentních jevů vyžaduje nesmírně jemnou síť a velmi malý časový krok. Náročnost takového výpočtu je ale tak obrovská, že jej reálně lze provést pouze u jednodušších případů a i u nich je to velmi obtížné.

Tento problém se v technické praxi řeší použitím zjednodušených modelů turbulence, které nezachycují veškeré detaily turbulentního proudění, ale dávají nám postačující informace pro většinu inženýrských aplikací.

2.3.2 Reynoldsovo středování

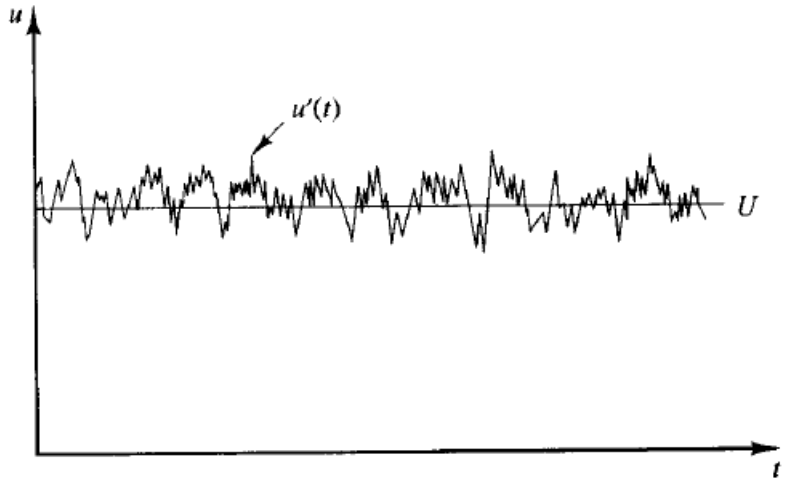
Základem většiny používaných modelů turbulence je Reynoldsovo středování, které rozděluje fyzikální veličinu na časově nezávislou střední hodnotu a fluktaci kolem této hodnoty.

$$\boldsymbol{\varphi}(\mathbf{x}, t) = \overline{\boldsymbol{\varphi}}(\mathbf{x}) + \boldsymbol{\varphi}'(\mathbf{x}, t) \tag{5}$$

$$\overline{\boldsymbol{\varphi}}(\mathbf{x}) = \lim_{T \to \infty} \frac{1}{T} \int_0^T \boldsymbol{\varphi}(\mathbf{x}, t) dt \tag{6}$$

$$\overline{\boldsymbol{\varphi}'}(\mathbf{x}, t) = 0 \tag{7}$$

Kde $\boldsymbol{\varphi}(\mathbf{x}, t)$ je fyzikální veličina obecně závislá na čase a materiální souřadnici. Grafické znázornění středování rychlosti proudění vypadá takto:



Obrázek 1: Rozdělení rychlosti proudění dle Reynoldsova středování [7]

Zde $\mathbf{U}$ představuje časově konstantní složku a $\mathbf{u}'(t)$ fluktaci kolem ní.

Po aplikaci tohoto principu na rovnice (3) a (4) pro rychlost $\mathbf{u}$ a tlak p dostaneme následující vztahy.

$$\mathbf{u} \cdot \nabla \mathbf{u} + \frac{1}{\rho} \nabla \overline{p} = \nabla \cdot (\nu \nabla \overline{\mathbf{u}} - \overline{\mathbf{u}'\mathbf{u}'}) \tag{8}$$

$$\nabla \overline{\mathbf{u}} = 0 \tag{9}$$

Problémem je poslední člen rovnice (8) $-\overline{\mathbf{u}'\mathbf{u}'}$, který nejsme schopni určit. Lze ho však nahradit na základě analogie s Newtonovým zákonem pro vazká napětí. Nazveme ho tedy Reynoldsovým napětím $\boldsymbol{\tau}^t$ a nahradíme ho dle následující rovnice.

$$\boldsymbol{\tau}^t = 2\nu_t \mathbf{S} \tag{10}$$

Kde $\mathbf{S}$ je tenzor rychlosti deformace

$$\mathbf{S} = \frac{1}{2}[\nabla \mathbf{u} + (\nabla \mathbf{u})^T] \tag{11}$$

a ν_t je kinematická turbulentní viskozita, která nám dává informaci o míře turbulence proudění a v jejímž výpočtu se odlišují jednotlivé modely.

2.3.3 Spalartův-Allmarasův model

Pro mnou řešený případ 2D obtékání profilu křídla jsem zvolil Spalartův-Allmarasův model turbulence. Ten byl vyvinut speciálně pro aerodynamické aplikace. Jedná se o jednorovnicový model, který přidává výpočet jen jedné proměné navíc, a tou je $\tilde{\nu}$. Díky tomu je méně náročný na využití paměti, než většina používaných modelů. Vykazuje také dobrou konvergenci a stabilitu. Z $\tilde{\nu}$ je poté vypočtena kinematická viskozita ν_t ($\nu_t = f(\tilde{\nu})$).

Rovnice, potřebné k samotnému výpočtu ν_t, podle tohoto modelu můžeme najít například v [6]. Jejich doplněním k (8), (9) dostaneme soustavu rovnic, kterou jsme schopni numericky řešit.

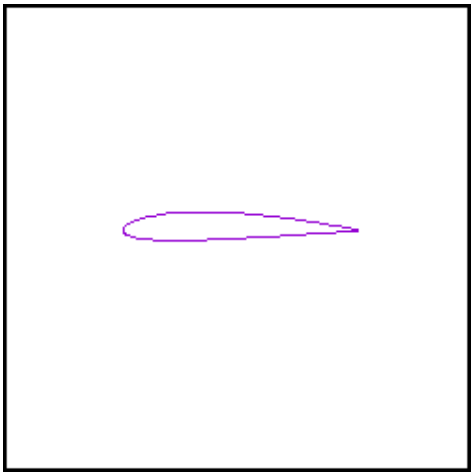
2.4 Metoda konečných objemů

K řešení soustavy parciálních diferenciálních rovnic, popsaných v předchozím textu, bude v další části této práce použit program OpenFOAM. Ten ji řeší pomocí metody konečných objemů. V té rozdělujeme danou oblast na konečný počet dílčích podoblastí, pro které jsme schopni numericky řešit příslušné diferenciální rovnice. Tím dostaneme soustavu lineárních algebraických rovnic, kterou řešíme opět pomocí vhodné numerické metody. Výsledný průběh fyzikálních veličin vypočtených touto metodou není spojitý, ale mění se skokově na hranici sousedních buněk, z čehož je jasné, že tvorba odpovídající sítě je základem správného výpočtu.

Úplný matematický popis metody konečných prvků přesahuje rámec této práce, proto zde uveden nebude, můžeme ho najít například v [2].

3 ZADÁNÍ ÚLOHY

Ve zbytku této práce bude popsán postup při 2D simulaci obtékání profilu křídla NACA 2412. Ta bude provedena pro několik úhlů náběhu v rozmezí $(-5, 5)°$. Ze získaných dat bude vyhodnocena závislost koeficientů vztlaku a odporu na úhlu náběhu. Geometrie úlohy vypadá následovně:



Obrázek 2: Geometrie úlohy

Levým okrajem vstupuje vzduch o rychlosti 30 $^{m}/_{s}$ a pravým vystupuje. Horní a dolní okraj zde nemá představovat reálné ohraničení oblasti, jelikož simulujeme obtékání křídla letadla v atmosféře. Toho bude docíleno později při tvorbě sítě a definování okrajových podmínek. Hustota vzduchu je $\rho = 1\ ^{kg}/_{m^3}$ a kinematická viskozita $\nu = 1 \cdot 10^{-5}\ ^{m^2}/_{s}$.

4 TVORBA SÍTĚ

4.1 Úvod

Jak jsem již zmiňoval v předešlých kapitolách, tvorba sítě je základem každé numerické simulace. Struktura, velikost a rozložení buněk, z kterých se tato síť skládá, zásadně ovlivňuje výsledek výpočtu. Čím menší tyto buňky budou, tím přesnějších výsledků dosáhneme. S přesností však roste také čas potřebný k provedení výpočtu. Snažíme se tedy sestrojit síť tak, aby nám zajistila potřebnou přesnost pro daný problém a zároveň umožnila realizovat výpočet v přijatelném čase. Toho lze nejlépe dosáhnout použitím jemné sítě v místech, která nás zajímají a kde předpokládáme vysoké hodnoty derivací sledovaných veličin a naopak sítě hrubé v místech méně důležitých z hlediska zkoumaných jevů. Toto rozdělení vychází z fyzikální podstaty problému a je zcela na posouzení osoby provádějící výpočet. V některých případech nemusíme být schopni dostatečně předpovědět průběhy veličin. V takové situaci je možné provést simulaci s hrubší sítí, kterou následně na základě výsledků zjemníme na místech, kde pozorujeme velké změny mezi sousedními buňkami. Tento postup opakujeme, dokud výsledky nejsou uspokojivé.

Programů na tvorbu sítě existuje mnoho, výběr záleží na našich potřebách a vkusu. Pro výpočet prováděný v této práci použiji program Gmsh. Ten představuje poměrně jednoduchý způsob tvorby sítě s možnostmi více než postačujícími pro daný případ. S tímto programem můžeme pracovat dvěma způsoby. Prvním z nich je využití grafického prostředí, které je jeho součástí. Druhým způsobem je tvorba skriptu skládajícího se z jednotlivých příkazů. Ten poté pouze otevřeme v programu a zobrazíme tak již hotovou síť. Ačkoliv první způsob je intuitivnější a jednodušší na naučení, v této práci použiji druhý zmíněný přístup. A to hlavně proto, že nám dává možnost algoritmizace tvorby sítě. Bez té bychom náš případ řešili velmi těžko, jelikož pro dostatečně přesné popsání tvaru profilu křídla budeme potřebovat alespoň 200 bodů. Navíc potřebujeme tuto síť vytvořit pro různé úhly náběhu. Tyto dvě skutečnosti naprosto vylučují použití grafického rozhraní, v kterém by pracnost zadání potřebných parametrů přesáhla všechny meze. Proto se v další části této práce budu zabývat psaním pro-

gramu, který dokáže vytvořit skript pro Gmsh se sítí pro libovolný 4-místný NACA profil.

4.2 Tvorba programu na přípravu sítě v Gmsh

K tvorbě tohoto programu je možné využít v podstatě libovolného programovacího jazyka. Já jsem se rozhodl pro objektově orientovaný jazyk $C++$, protože jsem s ním již dříve pracoval.

4.2.1 Struktura programu

Při psaní jakéhokoliv programu je vhodné si jako první krok rozvrhnout jeho strukturu. U objektově orientovaného programování to z velké části znamená návrh tříd, které budou v programu použity. Dobrým zvykem je tento návrh udělat co nejjednodušeji a nejobecněji, aby v případě potřeby nebyl problém program rozšířit či nějak upravit.

Vzhledem k těmto zásadám jsem provedl následující návrh tříd a funkcí, které bude program obsahovat.

```
struct coord{
    double x;
    double y;
};

class Polyline{
public:
    //number of grid points across length
    int gridpts;
    //number of points that polyline has
    virtual int size()const=0;
    //returns chosen point 0 to size()-1
    virtual coord point(int i)const=0;
};

class NACA:public Polyline{
public:
    //upper/lower airfoil line points
    coord* up;
    coord* down;
    //mean camber line points
    coord* line;
    //spline at the end of airfoil
    coord spline[9];
    int size()const;
    coord point(int i)const;
    NACA(int isize);
    ~NACA();
    //counts airfoil points out of parameters passed
    void CountPoints(double m,double p,double t);
};
```

Obrázek 3: Definice tříd

```
// creates file with profile coordinates called filename.txt
void toFile(const Polyline& p,string filename);
// creates gmesh script called filename.geo profile rotated by angle clockwise
void toGmsh(const Polyline& p,string filename, double angle);
// returns distance between two points passed
double dist(const coord a,const coord b);
```

Obrázek 4: Definice funkcí

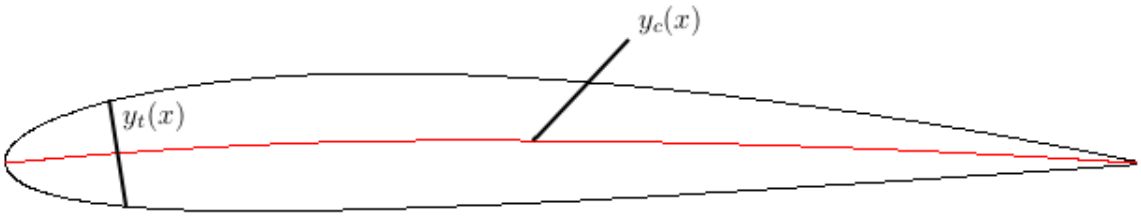
Na obrázku (3) vidíme že program obsahuje jednu strukturu a dvě třídy. Tato struktura

obsahuje dvě reálná čísla. V programu je používána na uchovávání souřadnic bodů. **NACA** je třída představující 4-místný NACA profil. Její konstruktor má jeden parametr, tím je počet bodů podél délky profilu. Podle tohoto čísla dynamicky naalokuje potřebnou paměť k uložení bodů střední, horní i spodní čáry popisující tvar křídla. Destruktor pak tuto pamět opět uvolní. Funkce **CountPoints** vypočte ze tří parametrů body tvořící profil. K výpočtu tvaru NACA profilů a jeho implementaci v této funkci se dostaneme později. Třída **Polyline**, ze které třída **NACA** dědí, je třídou virtuální, její instanci tedy nelze založit. Při aktuálním rozsahu programu se jeví jako zbytečná. Důvod, proč jsem ji takto zadefinoval, je výše zmíněné případné rozšíření programu, dostaneme se k němu později. Na obrázku (4) vidíme funkce **toFile** a **toGmsh**, z jejichž názvů je celkem jasné, co dělají. První z nich vytvoří textový soubor obsahující výpis souřadnic bodů tvořících profil. Druhá připraví již zmiňovaný skript s hotovou sítí kolem profilu pro Gmsh. Ani jedna z nich však nemá jako svůj parametr třídu **NACA**, ale referenci na třídu **Polyline**. Důvodem je možnost jednoduché implementace dalších typů profilů. V takovém případě by stačilo vytvořit novou třídu, dědící z **Polyline**, která by představovala jiný profil křídla. Pokud bychom do ní správně implementovali funkce size a point, mohli bychom poté jednoduše použít stejnou funkci **toGmsh** na vytvoření sítě i pro tento profil. Princip, který je zde použit, kdy můžeme zaměňovat třídy dědící ze stejné mateřské třídy, se nazývá polymorfismus a je jednou z velkých výhod objektového programování. Funkce dist je jen funkcí pomocnou, která je použita ve funkci toGmsh a není nutné se jí zabývat.

4.2.2 Výpočet tvaru NACA profilu

NACA profilů je více druhů, odlišují se množstvím parametrů nutných k jejich definování. V této práci budeme provádět simulaci obtékání kolem 4-místného profilu, zaměříme se tedy jen na jeho popis a implementaci výpočtu do tvořeného programu.

Tento profil je popsán funkcí střední křivky $y_c(x)$ a rozložením tloušťky podél ní $y_t(x)$. Tato tloušťka je přičtena na obě strany vždy kolmo ke křivce dle obrázku (5).



Obrázek 5: profil NACA 2412

Do těchto funkcí musíme dosadit tři parametry, nazveme je m, p a t. Pokud máme například profil NACA 2412, tak $m = 2$, $p = 4$ a $t = 12$. Každý z těchto parametrů má svůj význam, m je maximální prohnutí v procentech délky profilu, p je x-ová pozice tohoto prohnutí v desítkách procent délky profilu a t je maximum tloušťky profilu opět v procentech délky. V našem případě je tedy střední čára maximálně prohnutá o 2% délky, toto maximum je v 40% délky profilu a maximální tloušťka je 12% délky (ta je měřena jen na jednu stranu od střední čáry). Tento profil můžeme vidět na obrázku (5). Získáme ho dosazením do následujících vztahů:

$$y_c = \frac{m}{p^2} \cdot \left(2p \cdot \left(\frac{x}{c}\right) - \left(\frac{x}{c}\right)^2\right) \tag{12}$$

$$y_c = \frac{m}{(1-p)^2} \cdot \left((1-2p) + 2p\left(\frac{x}{c}\right) - \left(\frac{x}{c}\right)^2\right) \tag{13}$$

$$y_t = 5t \cdot \left(0.2969 \cdot \sqrt{\frac{x}{c}} - 0.1260\left(\frac{x}{c}\right) - 0.3516\left(\frac{x}{c}\right)^2 + 0.2843\left(\frac{x}{c}\right)^3 - 0.1015\left(\frac{x}{c}\right)^4\right) \tag{14}$$

$$\frac{dy_c}{dx} = \frac{2m}{cp^2} \cdot \left(p - \left(\frac{x}{c}\right)\right) \tag{15}$$

$$\frac{dy_c}{dx} = \frac{2m}{c(1-p)^2} \cdot \left(p - \left(\frac{x}{c}\right)\right) \tag{16}$$

$$\theta = arctg\left(\frac{dy_c}{dx}\right) \tag{17}$$

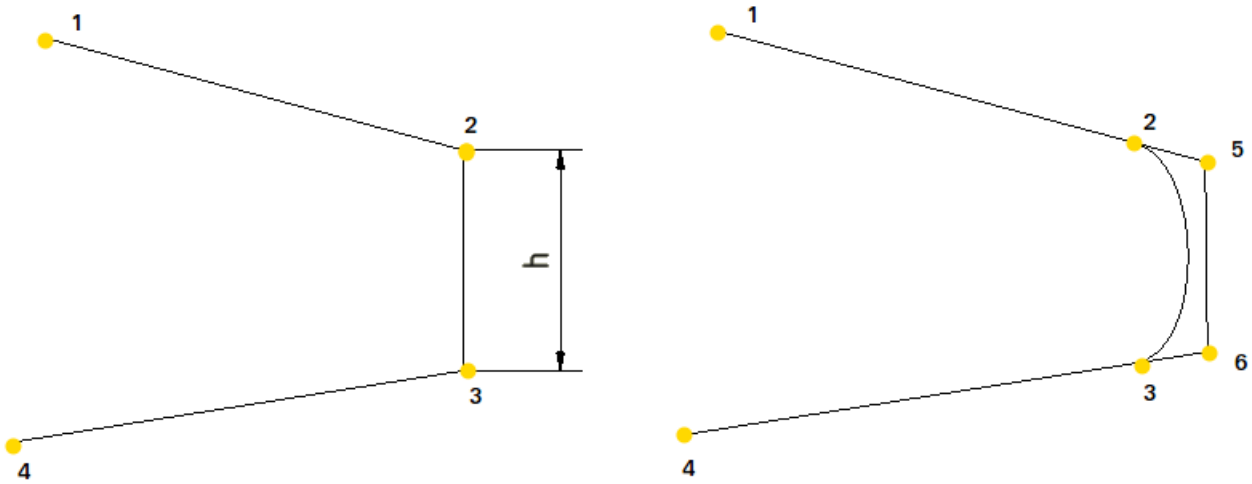
$$x_u = x - y_t \cdot sin\theta \tag{18}$$

$$x_l = x + y_t \cdot sin\theta \tag{19}$$

$$y_u = y_c + y_t \cdot cos\theta \tag{20}$$

$$y_l = y_c - y_t \cdot cos\theta \tag{21}$$

Kde c je délka profilu. Vztahy (12) a (15) platí pro x od 0 do místa maximálního prohnutí a vztahy (13) a (16) pro ostatní x. Souřadnice bodu na horním okraji profilu jsou $[x_u, y_u]$, na spodním $[x_l, y_l]$. Tloušťku y_t musíme k střední čáře profilu přičítat/odečítat přenásobenou $sin\theta$ či $cos\theta$, protože ji přičítáme/odečítáme kolmo k ní. Problémem tohoto popisu je odtoková hrana profilu. Koeficienty v závorce ve vztahu (14) nedávají dohromady nulu, pro $x = c$ tedy není tloušťka nulová. To způsobí vznik malé plošky na konci profilu. Možným řešením je mírně změnit jeden z koeficientů v (14) aby tloušťka pro $x = c$ byla nulová. V reálném případě však nebude pravdivý ani jediný z těchto případů. Tento problém jsem v našem případě vyřešil vytvořením Beziérovy křivky 3. stupně mezi posledními dvěmi body profilu, kterou můžeme vidět na obrázku (6).



Obrázek 6: Odtoková hrana profilu před a po úpravě

Zde řídící body Beziérovy křivky jsou 2,5,6 a 3, vzdálenost $|25| = |36| = h/3$. Její mate-

matické vyjádření pro 3. stupeň je následující [3].

$$\mathbf{P(t)} = B_{0,3}\mathbf{V_0} + B_{1,3}\mathbf{V_1} + B_{2,3}\mathbf{V_2} + B_{3,3}\mathbf{V_3} \tag{22}$$

$$B_{0,3} = (1-t)^3 \tag{23}$$

$$B_{1,3} = 3t \cdot (1-t)^2 \tag{24}$$

$$B_{2,3} = 3t^2 \cdot (1-t) \tag{25}$$

$$B_{3,3} = t^3 \tag{26}$$

$\mathbf{V_0} - \mathbf{V_3}$ jsou řídící body a $t \in \langle 0, 1\rangle$ je parametr vyjadřující v jaké části křivky se nacházíme, $\mathbf{P}(0) = \mathbf{V_0}$ a $\mathbf{P}(1) = \mathbf{V_3}$.

Nyní musíme výpočet profilu uvedený výše implementovat do tvořeného programu. Budeme se tedy zabývat funkcí **CountPoints**, která je součástí třídy **NACA**, její definice je na obrázku (3). Pro náš případ budeme uvažovat $c = 1$, tím pádem ho ve všech vztazích můžeme vynechat. Prvním problémem, který musíme řešit při výpočtu bodů profilu je jejich x-ové rozložení. Nejjednodušším řešením je zvolit rovnoměrné rozložení. To však není ani zdaleka ideálním řešením, protože na začátku a konci profilu se y mění mnohem rychleji v závislosti na x , než uprostřed profilu, kde je tvar křivky téměř lineární. Vzhledem k tomuto faktu jsem pro náš případ zvolil kosínové rozložení bodů dle rovnice (27).

$$x = \frac{1 - cos\beta}{2} \tag{27}$$

Kde $\beta \in \langle 0, \pi\rangle$, to nám zajistí malé Δx na obou koncích profilu a větší uprostřed. S tímto rozložením můžeme pomocí "for"cyklů postupně provést výpočty (12) až (21) a dopočítat body Beziérovy křivky na odtokové hraně, pro náš případ jsem jich zvolil 9. Celá funkce **CountPoints** je na obrázcích (7) a (8).

```
void NACA::CountPoints(double m, double p, double t){
    double* dy_dx,*theta,*yt;
    double h,d;
    coord s1,s2;//b-spline control points
    dy_dx=new double[gridpts];
    theta=new double[gridpts];
    yt=new double[gridpts];
    //camber gradient
    for (int i = 1; i <gridpts; i++) {
        //x point cosine distribution
        line[i].x=(1-cos((M_PI/(gridpts-1))*i))/2;
        // before max camber
        if(line[i].x>=0 && line[i].x<=p){
            //y point
            line[i].y=(m/pow(p,2))*((2*p*line[i].x)-pow(line[i].x,2));
            //gradient
            dy_dx[i]=(2*m/(pow(p,2)))*(p-(line[i].x));
        }
        //after max camber
        else  if(line[i].x>p && line[i].x<=1){
            //y point
            line[i].y=(m/pow(1-p,2))*((1-2*p)+(2*p*line[i].x)-pow(line[i].x,2));
            //gradient
            dy_dx[i]=(2*m/(pow(1-p,2)))*(p-(line[i].x));
        }
        theta[i]=atan(dy_dx[i]);
    }
    //thickness distribution
    for (int i = 1; i < gridpts; i++) {
        yt[i]=5*t*(0.2969*sqrt(line[i].x)-0.1260*(line[i].x)
                -0.3516*pow(line[i].x,2)+0.2843*pow(line[i].x,3)
                -0.1015*pow(line[i].x,4));
    }
    //upper and lower surface
    for (int i = 1; i < gridpts; i++) {
        //upper
        up[i].x=line[i].x-yt[i]*sin(theta[i]);
        up[i].y=line[i].y+yt[i]*cos(theta[i]);
        //lower
        down[i].x=line[i].x+yt[i]*sin(theta[i]);
        down[i].y=line[i].y-yt[i]*cos(theta[i]);
    }
```

Obrázek 7: Implementace funkce **CountPoints** část 1

```
    //b-spline at the end
        //control points
    h=dist(up[gridpts-1],down[gridpts-1])/3;
    s1.x=up[gridpts-1].x+((up[gridpts-1].x-up[gridpts-2].x)
            /dist(up[gridpts-1],up[gridpts-2]))*h;
    s1.y=up[gridpts-1].y+((up[gridpts-1].y-up[gridpts-2].y)
            /dist(up[gridpts-1],up[gridpts-2]))*h;
    s2.x=down[gridpts-1].x+((down[gridpts-1].x-down[gridpts-2].x)
            /dist(down[gridpts-1],down[gridpts-2]))*h;
    s2.y=down[gridpts-1].y+((down[gridpts-1].y-down[gridpts-2].y)
            /dist(down[gridpts-1],down[gridpts-2]))*h;
        //points on spline
    d=1./10.;
    for (int i = 0; i <9; i++) {
        spline[i].x=pow(1-d,3)*up[gridpts-1].x+3*d*pow(1-d,2)*s1.x
                +3*pow(d,2)*(1-d)*s2.x+pow(d,3)*down[gridpts-1].x;
        spline[i].y=pow(1-d,3)*up[gridpts-1].y+3*d*pow(1-d,2)*s1.y
                +3*pow(d,2)*(1-d)*s2.y+pow(d,3)*down[gridpts-1].y;
        d+=1./10.;
    }

    delete[] dy_dx;
    delete[] theta;
    delete[] yt;
}
```

Obrázek 8: Implementace funkce **CountPoints** část 2

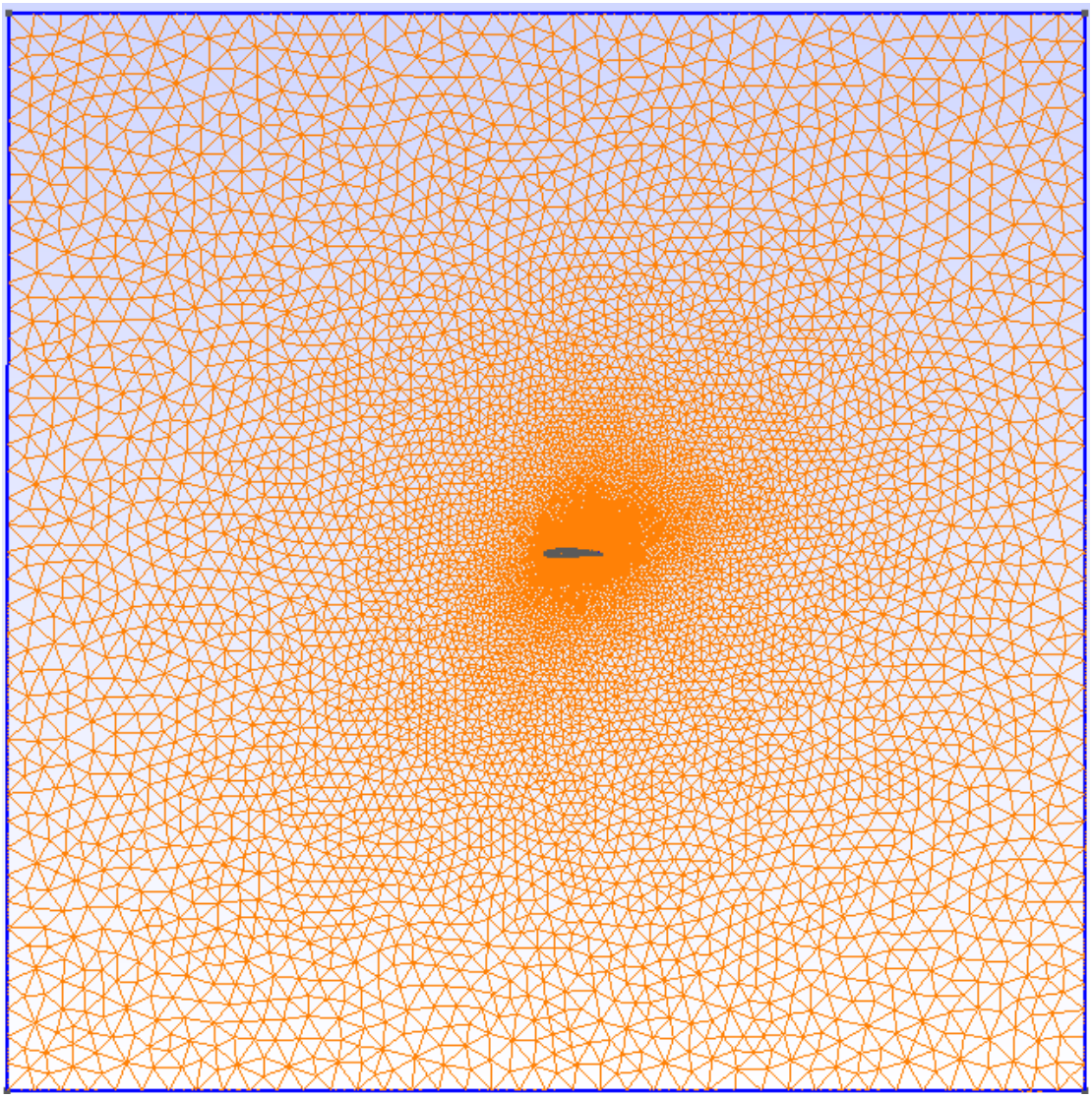
4.2.3 Tvorba skriptu pro Gmsh

Skript pro Gmsh je textový soubor obsahující jednotlivé příkazy. Ty jsou od sebe odděleny středníky. Při jeho otevření pomocí Gmsh jsou příkazy provedeny a zobrazí se nám jejich výsledek v grafické podobě. Jak již bylo zmíněno v kapitole 4.2.1, budeme tento skript tvořit pomocí našeho programu, konkrétně jeho funkce **toGmsh**. Než se však budeme zabývat její implementací, ukážeme si postup při tvorbě sítě v Gmsh a příkazy z kterých se skript bude skládat.

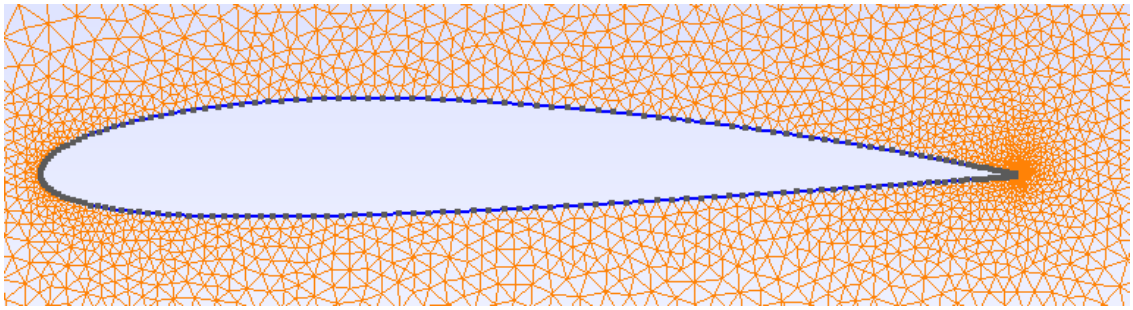
Při tvorbě sítě nejdříve musíme vymodelovat zadanou geometrii. Ta se bude v našem případě skládat z bodů pospojovaných do potřebného tvaru úsečkami. Gmsh nabízí i jiné křivky, my je však nebudeme potřebovat. Důvodem je fakt, že při převodu sítě z Gmsh do OpenFOAMu nebudou žádné křivky rozpoznány. Tvar, na kterém bude výpočet probíhat, bude tedy záviset jen na rozmístění bodů. Vzhledem k tomu musíme profil křídla popsat

dostatečným počtem bodů, který zajistí dobrou aproximaci jeho tvaru. Zvolil jsem tedy 100 bodů na horním i dolním okraji profilu. Spolu s 9 body na Beziérově křivce na odtokové hraně to dává 209 bodů popisujících celý profil. Bod vytvoříme příkazem *Point(n)*={*x,y,z,m*}. Zde n je identifikační číslo bodu, pomocí něj s ním můžeme dále pracovat. Žádne dva body ho tedy nesmí mít stejné. Tento způsob značení je použit u všech prvků, které v Gmsh definujeme. Unikátnost tohoto čísla musíme dodržovat jen v rámci jednoho prvku, můžeme tedy například mít bod a úsečku se stejným číslem. Parametry *x,y,z* jsou souřadnice profilu. Gmsh vždy pracuje s 3D prostorem, když tedy chceme vytvářet 2D geometrii budeme volit pro všechny body $z = 0$. Poslední parametr m je velikost buněk sítě v okolí bodu. Na obvodu jsem zvolil 0.5 a u bodů profilu 0.05, to zajistí jemnější síť v okolí profilu a hrubší na krajích výpočtové oblasti. Úsečku vytvoříme příkazem $Line(n) = \{n_{B1}, n_{B2}\}$. Kde n_{B1}, n_{B2} jsou identifikační čísla bodů, které chceme spojit. Pomocí těchto dvou příkazů jsme tedy schopni vytvořit profil křídla i čtvercovou oblast kolem něj. Jeho stranu jsem zvolil dvacetkrát větší než délku profilu. Tím potlačíme vliv okrajů na obtékání profilu, což je v našem případě žádoucí, protože se snažíme simulovat obtékání při letu v atmosféře. Pro náš případ bude tedy strana čtverce výpočtové oblasti dvacet.

V dalším kroku musíme určit plochu, která bude obsahovat síť. Abychom to mohli udělat, musíme nejdříve vytvořit z existujících čar uzavřené smyčky. Ty ohraničují plochu, kde chceme mít výpočtovou síť. Vytvoříme je pomocí příkazu $Line\ Loop(n) = \{n_{L1}, n_{L2}, \cdots n_{Lm}\}$. Kde n_{Lx} je identifikační číslo x-té úsečky. V našem případě budeme mít smyčky dvě. První bude čtvercová hranice výpočtové oblasti, druhá bude samotný profil křídla. Pak již můžeme zadefinovat oblast obsahující síť, to uděláme příkazem $Plane\ Surface(n) = \{n_{LL1}, n_{LL2}, \cdots n_{LLm}\}$. Kde n_{LLx} je x-tá smyčka, v našem případě budou pouze dvě, které jsem již zmiňoval. V tuto chvíli nám Gmsh automaticky vytvoří trojúhelníkovou nestrukturovanou síť, kterou vidíme na obrázku (9) a (10).



Obrázek 9: Nestrukturovaná trojúhelníková síť: celá oblast



Obrázek 10: Nestrukturovaná trojúhelníková síť: detail

Ta však není vhodná pro náš výpočet. Provedeme tedy několik úprav. Zaprvé změníme síť na čtvercovou, tím zmenšíme počet buněk na polovinu při stejném počtu bodů. To zhruba dvakrát zrychlí výpočet bez zhoršení jeho výsledku. Toho dosáhneme příkazem *Recombine* $Surface\{n_{S1}, n_{S2}, \cdots n_{Sm}\}$. Kde n_{Sx} je identifikační číslo x-té oblasti. V našem případě máme oblast jen jednu.

Druhou úpravou bude změna algoritmu, který Gmsh používá ke generování sítě. Automaticky zvolený algoritmus "Delaunay" nahradíme algoritmem "Frontal quad". Ten je sice pomalejší, ale zajistí nám větší kvalitu buněk. Ty také uspořádá do částečně strukturované sítě. V našem případě se bude snažit je udělat čtvercové s hranami rovnoběžnými k výpočtové oblasti. Vybereme ho příkazem $MeshAlgorithm\ Surface\{n_S\} = n_A$. Zde n_S je číslo síťované oblasti pro kterou měníme algoritmus a n_A je číslo algoritmu, který chceme zvolit. V našem případě "Frontal quad"vybereme číslem 8.

Poslední úpravou bude vytvoření jemné strukturováné sítě kolem profilu křídla. Z fyzikální podstaty obtékání tělesa víme, že se v jeho blízkosti bude tvořit mezní vrstva. V ní se rychlost výrazně mění s vzdáleností od profilu. Vytvoříme tedy v jeho blízkosti síť tak, aby buňky byly rovnoběžné s obrysem profilu, kdy se zvětšující se vzdáleností od profilu poroste velikost buněk. To uděláme vytvořením pole kolem profilu příkazem *Field[1] = BoundaryLayer*. *BoundaryLayer* je typ pole definovaný Gmsh. Poté musíme zadat několik parametrů popisujících toto pole. K parametrům přistupujeme pomocí příkazu *Field[1].Parametr=....* Prvním z nich je *EdgesList*, to je seznam všech křivek s kterými chceme rovnoběžné buňky vytvořit, v našem případě všechny úsečky tvořící obrys profilu křídla. Další parametry definují velikost buněk tvořících toto pole. Jsou jimi *hwall_n*, který určuje velikost buněk na povrchu, *thickness*, určující maximální tloušťku na okraji pole a *ratio*, určující kolikrát se zvětší následující buňka oproti předchozí. V našem případě volíme *thickness = 0.05* a *ratio = 1.1*. Volba *hwall_n* je největším problémem. Pro tuto velikosti první buňky se používá semiempirický vztah nazývaný univerzální rychlostní profil. Ten zavádí bezrozměrnou vzdálenost (odlehlost) místa od profilu. Podle ní pak volíme velikost buňek. Je definována následujícími

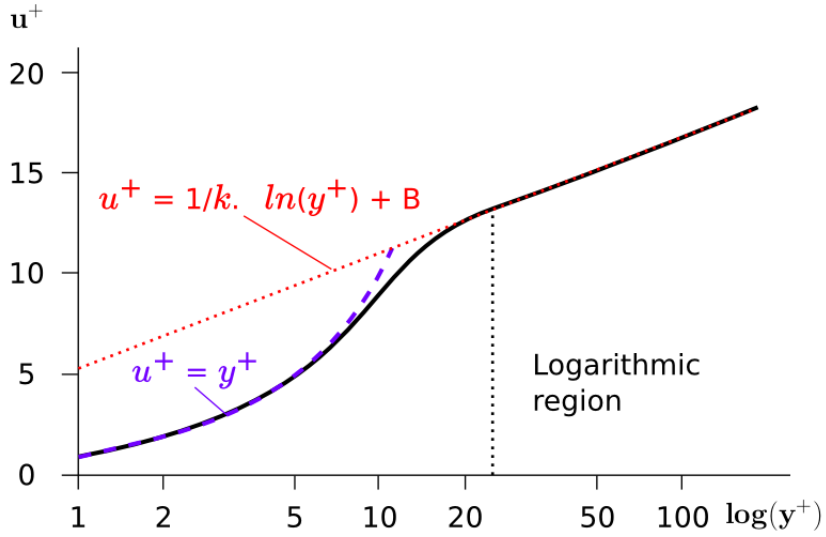
vztahy.

$$y^+ = \frac{y \cdot u_\tau}{\nu} \tag{28}$$

$$u_\tau = \sqrt{\nu \cdot \frac{\partial u}{\partial y}} \tag{29}$$

$$u^+ = \frac{u}{u_\infty} \tag{30}$$

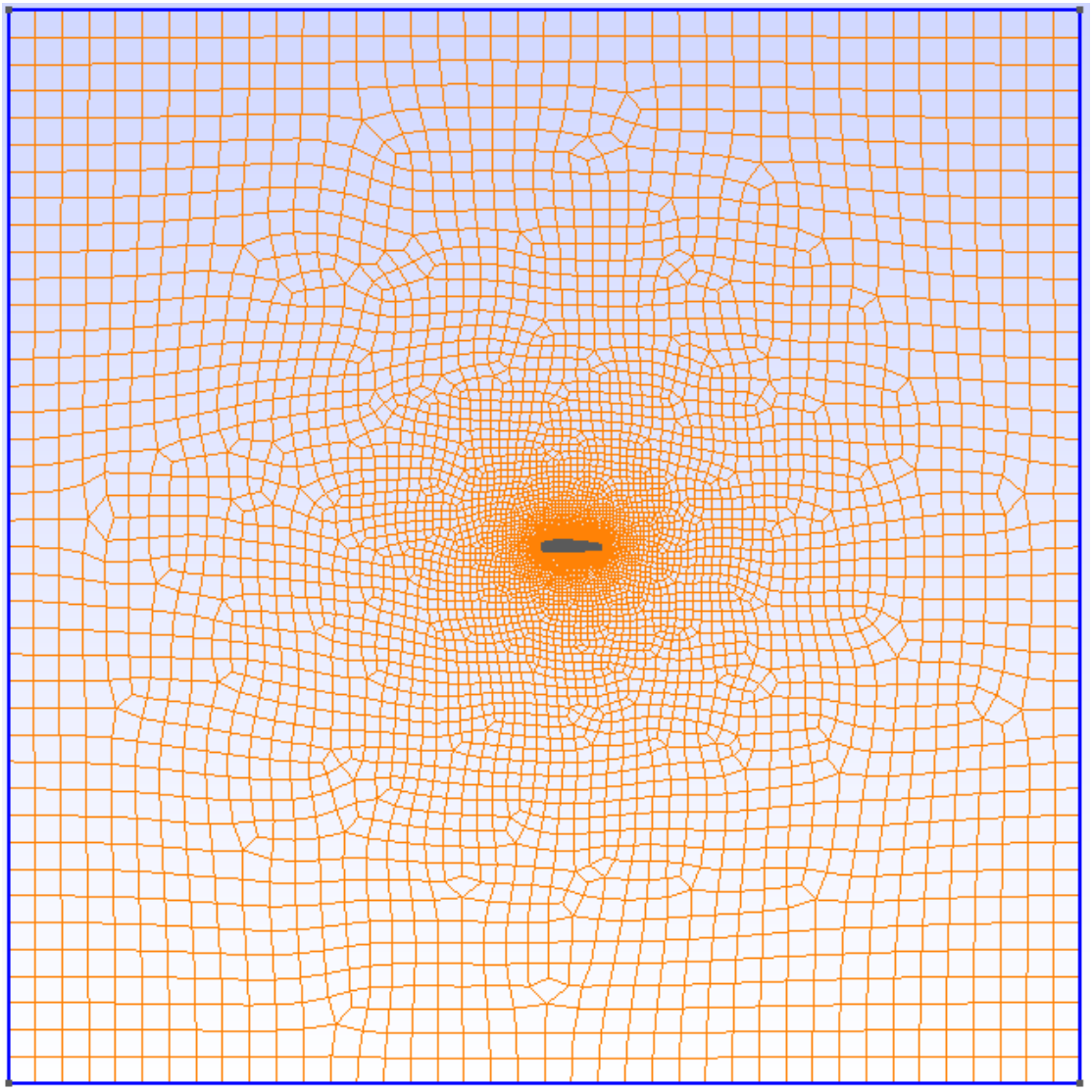
Kde y^+ je zmiňovaná bezrozměrná vzdálenost od stěny, u_τ je třecí rychlost a u^+ je bezrozměrná rychlost proudění. Závislost u^+ na y^+ pak vypadá podle obrázku (11).



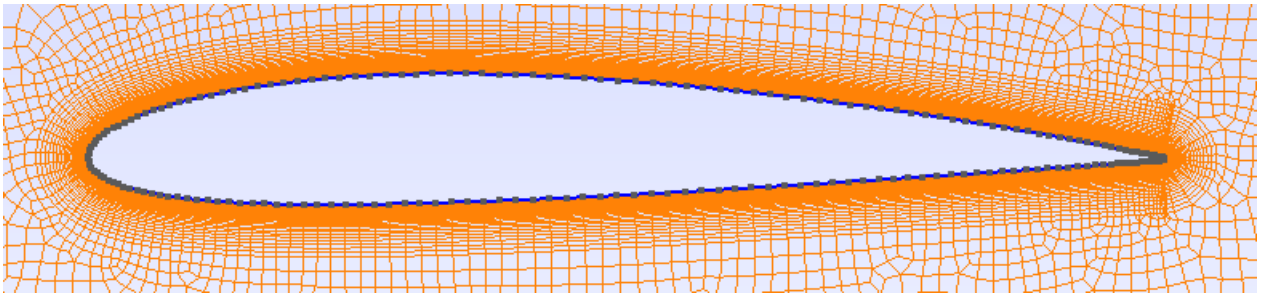
Obrázek 11: Univerzální rychlostní profil

Oblast, do které se chceme trefit první buňkou naší sítě, se liší podle volby krajových podmínek. V této práci nebudeme používat stěnové funkce. Proto se musíme první buňkou trefit do oblasti lineární závislosti na obrázku (11). Budeme se tedy snažit, aby y^+ první buňky bylo přibližně rovno jedné. Předpovědět vzdálenost při které toho dosáhneme před provedením výpočtu není možné, jelikož neznáme třecí rychlost. Je tedy vhodné ji odhadnout a po prvním provedení výpočtu zkontrolovat a případně síť upravit. Já jsem v této práci zvolil pro první výpočet 10^{-6}, to se ukázalo jako příliš malá hodnota. Zvětšil jsem ji tedy na 10^{-5}, při této hodnotě se $y^+ \approx 1$.

Finální podobu sítě po těchto úpravách vidíme na obrázcích (12) a (13).



Obrázek 12: Konečná podoba sítě: celá oblast



Obrázek 13: Konečná podoba sítě: detail

V další části této kapitoly se budeme zabývat generováním skriptu popsaného výše, což je hlavním úkolem tvořeného programu. Konkrétně tedy budeme řešit implementaci funkce **toGmsh**.

Její definici vidíme na obrázku (4). Je typu void, jelikož jejím cílem je zapsat potřebné věci do souboru a nepotřebuje tedy vracet žádnou proměnnou. K tomu, abychom mohli skript vytvořit, potřebujeme tři parametry. Těmi jsou jméno souboru ,do kterého skript napíšeme, úhel, o který chceme profil natočit, a třída **Polyline**, která obsahuje body profilu. Ta je zde předána jako konstantní reference. Reference proto, že místo kopírování celé třídy zkopírujeme pouze ukazatel na ní. Ten zabírá podstatně méně paměti a je to tedy efektivnější způsob. Tím že ji uděláme konstantní zabráníme tomu, abychom omylem přepsali její obsah.

U celého programu jsem se snažil o co nejobecnější funkčnost, aby v případě potřeby cokoliv změnit nenastal problém. Z tohoto důvodu je funkce psaná tak, aby byla schopna vytvořit skript pro jakoukoliv uzavřenou křivku popsanou třídou **Polyline**. Nevíme tedy předem ani počet bodů, které obsahuje. Tuto informaci nám dává funkce **size**, která je součástí **Polyline**. Přístup k jednotlivým bodům nám umožňuje funkce **point**, ta nám vrátí souřadnice bodu na základě jeho čísla. Pomocí těchto dvou funkcí není problém vytvořit "for"cyklus, který projede všechny tyto body a zapíše jejich souřadnice do souboru v potřebné syntaxi. Problém však nastává při řešení identifikačních čísel bodů a čar a při jejich správném použití v dalších příkazech. Tento problém jsem vyřešil zavedením několika proměnných, které počítají, kolik bylo jakých prvků vytvořeno, případně použito pro tvorbu jiného prvku. Jejich použití uvidíme na následujících obrázcích.

```
void toGmsh(const Polyline& p,string filename, double angle){
    ofstream file;
    double boundaryMeshSize=0.5;
    double meshsize=0.05;
    coord rotpoint,a,b;
    rotpoint.x=0.25;
    rotpoint.y=0;
    angle=(-angle/180)*M_PI;
    int point=0,line=0,linepoint=0,lineloop=0
    ,loopedlines=0,layerpoint=4;
    stringstream ss;
    string name;
    ss<<"./"<<filename<<".geo";
    name=ss.str();
    file.open(name.c_str(),ios::out);
    if(!file.is_open()){
        cout<<"error opening gmsh file"<<endl;
    }
```

Obrázek 14: Funkce **toGmsh** část 1

Na obrázku (14) vidíme definici proměnných, které budeme používat v dalších částech funkce a otevření souboru, do kterého budeme skript psát. Zde proměnné typu "int"jsou výše zmíněné proměnné, které nám budou dávat informaci o počtu vytvořených a použitých prvků v jednotlivých příkazech.

```
//square boundary
    //points
file<<"Point("<<++point<<") = {"<<-10<<","
        ""<<-10<<","<<0<<","<<boundaryMeshSize<<"};"<<endl;
file<<"Point("<<++point<<") = {"<<10<<","
        <<-10<<","<<0<<","<<boundaryMeshSize<<"};"<<endl;
file<<"Point("<<++point<<") = {"<<10<<","
        <<10<<","<<0<<","<<boundaryMeshSize<<"};"<<endl;
file<<"Point("<<++point<<") = {"<<-10<<","
        <<10<<","<<0<<","<<boundaryMeshSize<<"};"<<endl;
    //lines
for (int i = 0; i < 4; i++) {
    if(i==3){
        file<<"Line("<<++line<<") = {"<<++linepoint<<
                ","<<1<<"};"<<endl;
    }
    else{
        file<<"Line("<<++line<<") = {"<<++linepoint;
        file<<","<<linepoint+1<<"};"<<endl;
    }
}
    //loop
file<<"Line Loop("<<++lineloop<<") = {1,2,3,4};"<<endl;
loopedlines=4;
```

Obrázek 15: Funkce **toGmsh** část 2

V další části této funkce vidíme zápis příkazů tvořících čtvercovou hranici výpočtové oblasti. Ta bude vždy vypadat stejně. Vidíme zde první použití proměnných počítajících vytvořené body a úsečky. Tímto použitím ve tvaru *++"jméno proměnné"* zajistíme, že žádné dva prvky nebudou mít stejné identifikační číslo.

```
//profile
    //points
for (int i = 0; i < p.size(); i++) {
    a.x=p.point(i).x-rotpoint.x;
    a.y=p.point(i).y-rotpoint.y;
    b.x=a.x*cos(angle)-a.y*sin(angle);
    b.y=a.x*sin(angle)+a.y*cos(angle);
    file<<"Point("<<++point<<") = {"<<rotpoint.x+b.x<<","
            <<rotpoint.y+b.y<<","<<0<<","<<meshsize<<"};"<<endl;
}
    //lines
for (int i = 0; i < p.size()-1; i++) {
        file<<"Line("<<++line<<") = {"<<++linepoint;
        file<<","<<linepoint+1<<"};"<<endl;
}
file<<"Line("<<++line<<") = {"<<++linepoint<<",";
file<<linepoint-(p.size()-1)<<"};"<<endl;
    //loop
file<<"Line Loop("<<++lineloop<<") = {";
for (int i = loopedlines+1; i < line; i++) {
    file<<++loopedlines<<",";
}
file<<line<<"};"<<endl;
```

Obrázek 16: Funkce **toGmsh** část 3

Na obrázku (16) vidíme část funkce, která vytváří profil křídla. První cyklus vytvoří body, ty však nejprve otočí o úhel definovaný proměnnou *angle* kolem bodu *rotpoint*. Druhý cyklus vytvoří úsečky spojující body profilu. V něm vidíme druhé použití počítacích proměnných a to určení, které body byly již spojeny úsečkou. To zde ukazuje proměnná *linepoint*. Poslední úsečkou musíme spojit první s posledním bodem, uděláme to tedy až za cyklem. Třetím cyklem tvoříme smyčku definující profil křídla. Zde proměnná *loopedlines* opět slouží k počítání již použitých čar.

```
    //surface
    file<<"Plane Surface(1) = {1,2};"<<endl;
    //boundary layers
    file<<"Field[1] = BoundaryLayer;"<<endl;
    file<<"Field[1].EdgesList = {";
    for (int i = 0; i < p.size()-1; i++) {
        file<<++layerpoint<<",";
    }
    file<<++layerpoint<<"};"<<endl;
    file<<"Field[1].hwall_n = 0.00001;"<<endl;
    file<<"Field[1].thickness = 0.05;"<<endl;
    file<<"Field[1].ratio = 1.1;"<<endl;
    file<<"BoundaryLayer Field = 1;"<<endl;
    //making mesh quad
    file<<"Recombine Surface{1};"<<endl;
    file<<"MeshAlgorithm Surface{1}=8;"<<endl;
    file<<"Extrude {0, 0, 1} {"<<endl;
    file<<"Surface{1};"<<endl;
    file<<"Layers{1};"<<endl;
    file<<"Recombine;}"<<endl;

    file.close();
}
```

Obrázek 17: Funkce **toGmsh** část 4

V poslední části funkce zapisujeme do skriptu zbytek příkazů na úpravu sítě. Kromě jednoho cyklu na určení všech křivek, kolem kterých chceme utvořit pole zmiňované v první části této kapitoly, zde již všechny příkazy budou vypadat pro jakoukoli křivku stejně. Není tedy problém je do souboru napsat. Poslední čtyři řádky nebyly zmíněny. Ty jsou zde přidány proto, že při převodu z Gmsh do OpenFOAMu musí síť být 3D. Těmito příkazy vysuneme celou síť o jednu buňku. Vyhovíme tím tedy požadavku na 3D síť, aniž bychom změnili podstatu příkladu.

4.3 Uložení a převod sítě

Hotovou síť uložíme ve formátu **.msh*. To uděláme jednoduše v Gmsh vybráním možnosti *Save Mesh*. Tento soubor pak pomocí aplikace *gmshToFoam* převedeme do formátu, se kterým OpenFOAM umí pracovat.

5 VÝPOČET

Pokud jsme spokojeni s vytvořenou sítí, můžeme přejít k výpočtu. Před jeho spuštěním je potřeba nastavit výpočetní algoritmus, počáteční a okrajové podmínky, fyzikální vlastnosti tekutiny, případně využít některé z rozšiřujících funkcí. V programu OpenFOAM jsou všechny složky a soubory potřebné k definici sítě a spuštění výpočtu uloženy do jedné základní složky. V našem případě budeme provádět výpočet se stejným nastavením pro více úhlů náběhu. Vzhledem k tomu je vhodné vytvořit si jednu složku, ve které provedeme všechna nastavení, a tu poté kopírovat a měnit jen používanou síť.

5.1 Nastavení okrajových a počátečních podmínek

Při nastavování okrajových a počátečních podmínek musíme nejprve zadefinovat a pojmenovat okrajové plochy. K tomu nám poslouží aplikace *autoPatch* a *createPatch*, které jsou součástí OpenFOAM. První z nich rozdělí povrch geometrie na jednotlivé plochy, které mezi sebou svírají alespoň daný úhel. V našem případě jsou všechny hrany pod úhlem 90^o, takže můžeme pro *autoPatch* zvolit libovolný úhel $(0; 90)^0$. Druhá aplikace *createPatch* slouží k přejmenování a nastavení typu automaticky vygenerovaných okrajových ploch. K jejímu použití musíme nejprve zjistit, jak *autoPatch* rozdělí konkrétní geometrii. Podle toho poté vytvoříme soubor *system/createPatchDict*, který obsahuje přiřazení jmen a typů okrajových podmínek jednotlivým plochám. Použití této aplikace je tedy vhodné pro automatizaci pojmenovávání okrajových ploch v případě, že budeme výpočet opakovat pro různé sítě. Konkrétní soubor s nastavením pro náš případ je na obrázku (18).

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      createPatchDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

pointSync false;

patches
(
    {
        name airfoil;
        patchInfo { type wall; }
        constructFrom patches;
        patches (auto0);
    }
    {
        name empty_back;
        patchInfo { type empty; }
        constructFrom patches;
        patches (auto1);
    }
    {
        name empty_front;
        patchInfo { type empty; }
        constructFrom patches;
        patches (auto2);
    }
    {
        name outlet;
        patchInfo { type patch; }
        constructFrom patches;
        patches (auto3);
    }
    {
        name wall_down;
        patchInfo { type wall; }
        constructFrom patches;
        patches (auto4);
    }
    {
        name wall_up;
        patchInfo { type wall; }
        constructFrom patches;
        patches (auto5);
    }
    {
        name inlet;
        patchInfo { type patch; }
        constructFrom patches;
        patches (auto6);
    }
);

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

Obrázek 18: Soubor *system/createPatchDict*

Po určení jednotlivých okrajových ploch můžeme přistoupit k samotnému nastavení okra-

jových a počátečních podmínek. Toto nastavení provedeme ve složce *0*, která je součástí základní složky výpočtu. V našem případě, při použití Spalartova-Allmarasova modelu turbulence, v ní budou soubory *p, U, nut, nuTilda*. Pro všechny veličiny bude u přední a zadní stěny (*empty_front, empty_back*) zvolen typ *empty*. To zajistí, že rovnice nebudou řešeny v tomto směru a budeme tedy počítat 2D proudění.

Počáteční hodnota rychlosti v celé oblasti bude rovna $30^{m}/_{s}$ ve směru x, stejnou hodnotu budou mít i okrajové podmínky pro levou a pravou stěnu (*inlet, outlet*). Na povrchu křídla (*airfoil*) bude podmínka *noSlip*, která zajistí nulovou rychlost. Horní a dolní okraj (*wall_up, wall_down*) nemá představovat reálné ohraničení oblasti, proto zde byla použita podmínka *slip*. Ta zajistí, že tyto okraje nebudou nijak ovlivňovat výsledek výpočtu. Konkrétní podoba souboru *0/U* je na obrázku (19).

Počáteční hodnotu tlaku jsem zvolil 0. To způsobí, že výsledkem našeho výpočtu bude přetlak vůči tlaku atmosférickému. Pro okrajové podmínky na povrchu křídla i na horní a dolní stěně jsem použil podmínku nulového gradientu (*zeroGradient*). Pro vstup a výstup proudící tekutiny jsem použil podmínku pro tlak ve volně proudící tekutině (*freestreamPressure*). Soubor *0/p* je na obrázku(20).

Soubory *0/nut, 0/nuTilda* odpovídají veličinám ν_t a $\tilde{\nu}$ ze Spalartova-Allmarasova modelu turbulence (kap. 2.3.3). Počáteční hodnoty těchto veličin se volí částečně libovolně, abychom dosáhli odpovídajících výsledků, je dobré se řídit doporučenými hodnotami. Ty můžeme najít například v [6]. V našem případě jsem použil $\nu_t = \nu$ a $\tilde{\nu} = 5 \cdot \nu_t$.

Pro ν_t jsem tedy zvolil jako počáteční hodnotu $1 \cdot 10^{-5}\ {}^{m^2}/_{s}$. U okrajových podmínek na stěnách můžeme použít buď stěnové funkce nebo předepsat nulovou hodnotu. To záleží na jemnosti použité sítě v blízkosti stěn. V této práci jsem vytvořil dostatečně jemnou síť a použiji tedy okrajovou podmínku určující fixní hodnotu, jak jsem již zmiňoval v kapitole 4.2.3. Soubor pro $\tilde{\nu}$ je stejný jako pro ν_t. Jediným rozdílem je již zmiňovaná počáteční hodnota, ta je zde $5 \cdot 10^{-5}\ {}^{m^2}/_{s}$. Oba soubory můžeme vidět na obrázcích (21) a (22).

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    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 (30.00 0 0);

boundaryField
{
    airfoil
    {
        type             noSlip;
    }
    inlet
    {
        type            freestream;
        freestreamValue $internalField;
    }

    outlet
    {
        type            freestream;
        freestreamValue $internalField;
    }

    wall_down
    {
        type            slip;
    }
    wall_up
    {
        type            slip;
    }


    empty_back
    {
        type            empty;
    }
    empty_front
    {
        type            empty;
    }

}

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

Obrázek 19: Soubor *0/U*

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    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
{
    airfoil
    {
        type            zeroGradient;
    }
    inlet
    {
        type            freestreamPressure;
    }

    outlet
    {
        type            freestreamPressure;
    }

    wall_up
    {
        type            zeroGradient;
    }
    wall_down
    {
        type            zeroGradient;
    }


    empty_back
    {
        type            empty;
    }
    empty_front
    {
        type            empty;
    }

}

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

Obrázek 20: Soubor *0/p*

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nut;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 1e-05;

boundaryField
{
    airfoil
    {
        type            fixedValue;
        value           uniform 0;
    }
    inlet
    {
        type            freestream;
        freestreamValue $internalField;
    }

    outlet
    {
        type            freestream;
        freestreamValue $internalField;
    }

    wall_up
    {
        type            zeroGradient;
        value           uniform 0;
    }
    wall_down
    {
        type            zeroGradient;
        value           uniform 0;
    }


    emty_back
    {
        type            empty;
    }
    emty_front
    {
        type            empty;
    }

}

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

Obrázek 21: Soubor *0/nut*

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nuTilda;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 5e-05;

boundaryField
{
    airfoil
    {
        type            fixedValue;
        value           uniform 0;
    }
    inlet
    {
        type            freestream;
        freestreamValue $internalField;
    }

    outlet
    {
        type            freestream;
        freestreamValue $internalField;
    }

    wall_up
    {
        type            fixedValue;
        value           uniform 0;
    }
    wall_down
    {
        type            fixedValue;
        value           uniform 0;
    }


    empty_back
    {
        type            empty;
    }
    empty_front
    {
        type            empty;
    }

}

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

Obrázek 22: Soubor *0/nuTilda*

5.2 Nastavení fyzikálních vlastností

Fyzikální parametry potřebné k popsání chování tekutiny jsou hustota ρ, kinematická viskozita ν a reologický model. V našem případě zkoumáme proudění vzduchu, proto bude $\rho = 1\ {}^{kg}/_{m^3}$, $\nu = 1 \cdot 10^{-5}\ {}^{m^2}/_{s}$ a reologický model bude newtonský. Nastavení těchto vlastností provedeme v souboru *constant/transportPropeties*, jeho obsah je na obrázku (23).

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

transportModel  Newtonian;

rho             [1 -3 0 0 0 0 0] 1;

nu              [0 2 -1 0 0 0 0] 1e-05;

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

Obrázek 23: Soubor *0/transportProperties*

5.3 Nastavení parametrů výpočtu

Dalším nastavením, které je nutné provést před spuštěním výpočtu, je nastavení přesnosti, použitých numerických schémat, počtu iterací a ostatních parametrů potřebných pro výpočet. To provedeme v souborech *fvSchemes, fvSolution* a *controlDict*, které se nacházejí ve složce *system*.

Soubor *fvSchemes* obsahuje nastavení numerických schémat pro řešení jednotlivých členů parciálních diferenciálních rovnic. Těmi jsou: časová derivace, gradient, divergence a Laplaceův operátor. Dále zde najdeme nastavení interpolace hodnot mezi stěnami buňky, výpočtu složky gradientu kolmé na stěnu buňky a vzdálenosti od stěny. V našem případe bylo pro diskretizaci konvektivních členu využito schéma typu upwind s linearní interpolací s limiterem.

Difuzivní členy byly stejně jako Poissonova rovnice pro tlak diskretizovány pomocí schématu druhého řádu přesnosti. Podrobnosti viz [1].

V souboru *fvSolution* se nachází nastavení parametrů jednotlivých numerických řešičů. To jsou například celková přesnost výpočtu, relativní přesnost mezi iteracemi, relaxační koeficienty a další. Pro nás je nejdůležitější požadovaná přesnost výpočtu, pokud jí dosáhneme u všech veličin, výpočet se ukončí. Pro náš případ jsem zvolil pro tlak přesnost 10^{-6} a pro rychlost a $\tilde{\nu}$ 10^{-8}. Obsah souboru je na obrázku (24).

```
solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0.0001;
        smoother        GaussSeidel;
    }

    U
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-08;
        relTol          0.0001;
    }

    nuTilda
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-08;
        relTol          0.0001;
    }
}
SIMPLE
{
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;

    residualControl
    {
        p               1e-6;
        U               1e-6;
        nuTilda         1e-6;
    }
}

relaxationFactors
{
    fields
    {
        p               0.3;
    }
    equations
    {
        U               0.7;
        nuTilda         0.7;
    }
}

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

Obrázek 24: Soubor *system/fvSolution*

Soubor *controlDict* slouží k nastavení maximálního počtu iterací (pro případ, že nedosáhneme požadované přesnosti), časového kroku, intervalu výpisu výsledků do souboru, použitého řešiče a případně dalších přídavných funkcí OpenFOAMu. Základní nastavení parametrů výpočtu je na obrázku (25).

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

application     simpleFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         5000;

deltaT          1;

writeControl    timeStep;

writeInterval   200;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

Obrázek 25: Soubor *system/controlDict* část 1

Jak již bylo uvedeno v kapitole 3, jedním z hlavních cílů výpočtu je zjistit závislost koeficientů vztlaku a odporu na úhlu náběhu. Tyto koeficienty lze v OpenFOAMu jednoduše vypočíst pomocí přídavné funkce **forcesCoeffs**. Její použití a potřebné parametry nastavíme v druhé části souboru *controlDict*, která je na obrázku(26). Jako referenční plocha se u křídel volí plocha desky stejné velikosti jakou má křídlo. V našem případě je profil křídla dlouhý 1 a šířka, o kterou jsme profil vytáhli kvůli požadavku na 3D síť, je také rovna 1, referenční plocha tedy bude 1.

```
functions
{
forcesCoeffs
{
type forceCoeffs;
functionObjectLibs ( "libforces.so" ); // lib to load
outputControl timeStep;
outputInterval 100;
patches
(
airfoil
);
// name of fields
pName p;
UName U;
log true;
rho rhoInf;
rhoInf 1;
CofR ( 0 0 0 );
liftDir ( 0 1 0 );
dragDir ( 1 0 0 );
pitchAxis ( 0 0 1 );
magUInf 30.00;
lRef 0.305;
Aref 1;
}
}
// ************************************************************************* //
```

Obrázek 26: Soubor *system/controlDict* část 2

5.4 Nastavení modelu turbulence

Nastavení použitého přístupu k řešení turbulentního proudění najdeme v souboru *constant/turbulenceProperties*. Máme na výběr tři základní přístupy k modelování turbulence. Těmi jsou: **laminar** (bez modelování turbulence), **RAS** (modely založené na Reynoldsově středování) a **LES** (simulování velkých vírů). V této práci jsem vybral model typu **RAS** a to model Spalartův-Allmarasův, který je stručně popsán v kapitole 2.3.3.

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

simulationType RAS;

RAS
{
    RASModel        SpalartAllmaras;

    turbulence      on;

    printCoeffs     on;
}

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

Obrázek 27: Soubor *constant/transportProperties*

5.5 Spuštění výpočtu

Při přípravě výpočtu v této práci jsem používal verzi OpenFOAMu, která využívá ke svému spuštění program **Docker**. V této verzi stačí pro spuštění výpočtu pouze zadat název řešiče, který používáme. V našem případě je to tedy příkaz **simpleFoam**.

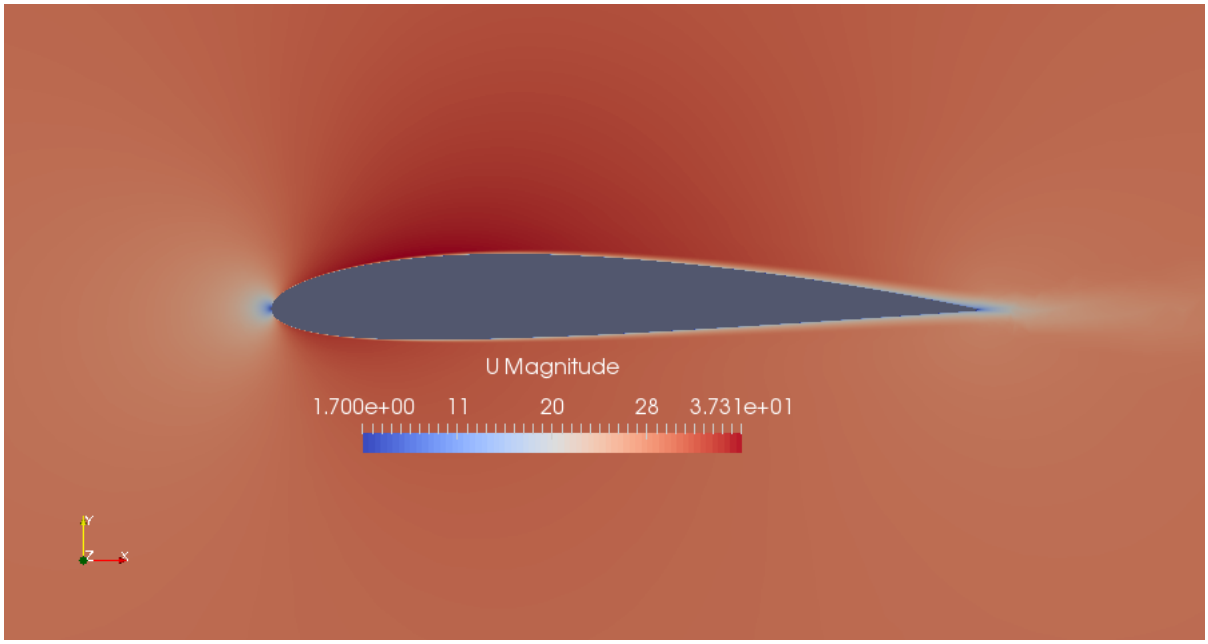
6 VÝSLEDKY VÝPOČTU

Po úspěšném provedení výpočtu můžeme přejít k zobrazení a vyhodnocení výsledků. V rámci této práce jsem provedl výpočet pro úhel náběhu $\langle -5, 5 \rangle^\circ$ vždy po půl stupni. Prezentovat výsledné průběhy veličin pro všechny tyto úhly by bylo velmi dlouhé, proto jsem vybral pouze tři úhly náběhu. Ostatní simulace se promítnou pouze do grafu závislosti koeficientu vztlaku a odporu na úhlu náběhu. Všechny průběhy vypočtených veličin budou zobrazeny pouze v okolí profilu křídla, protože zbytek oblasti není vzhledem k podstatě výpočtu důležitý.

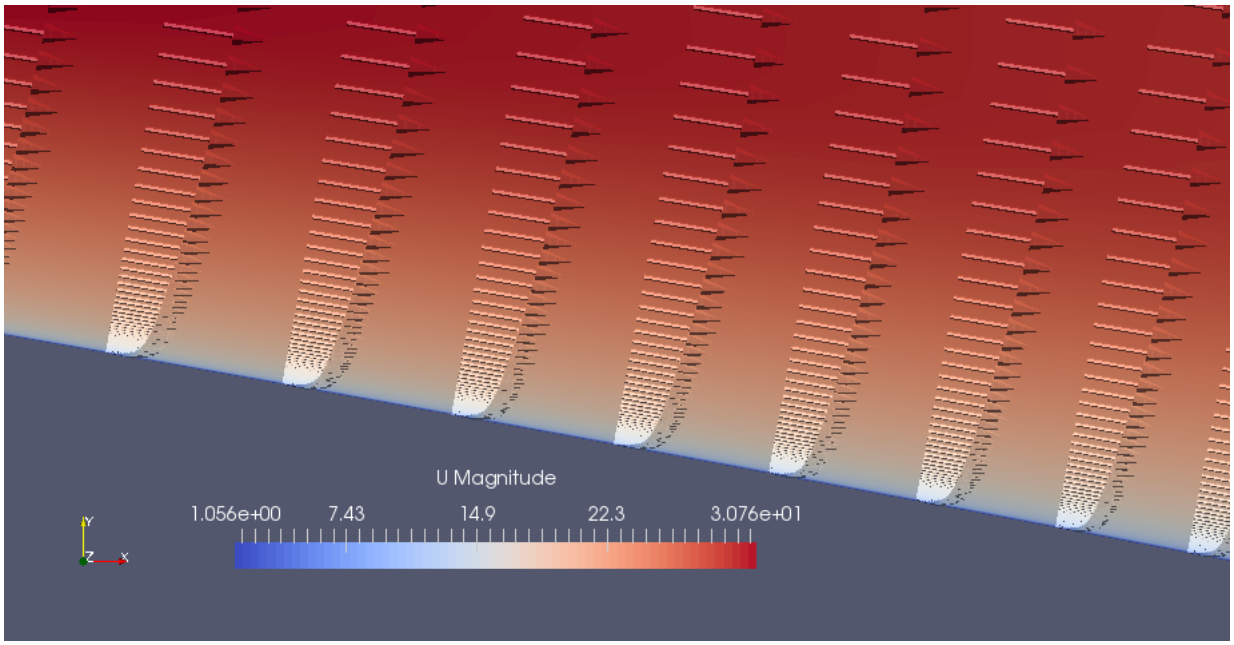
6.1 Prezentace výsledků

6.1.1 Úhel náběhu 0º

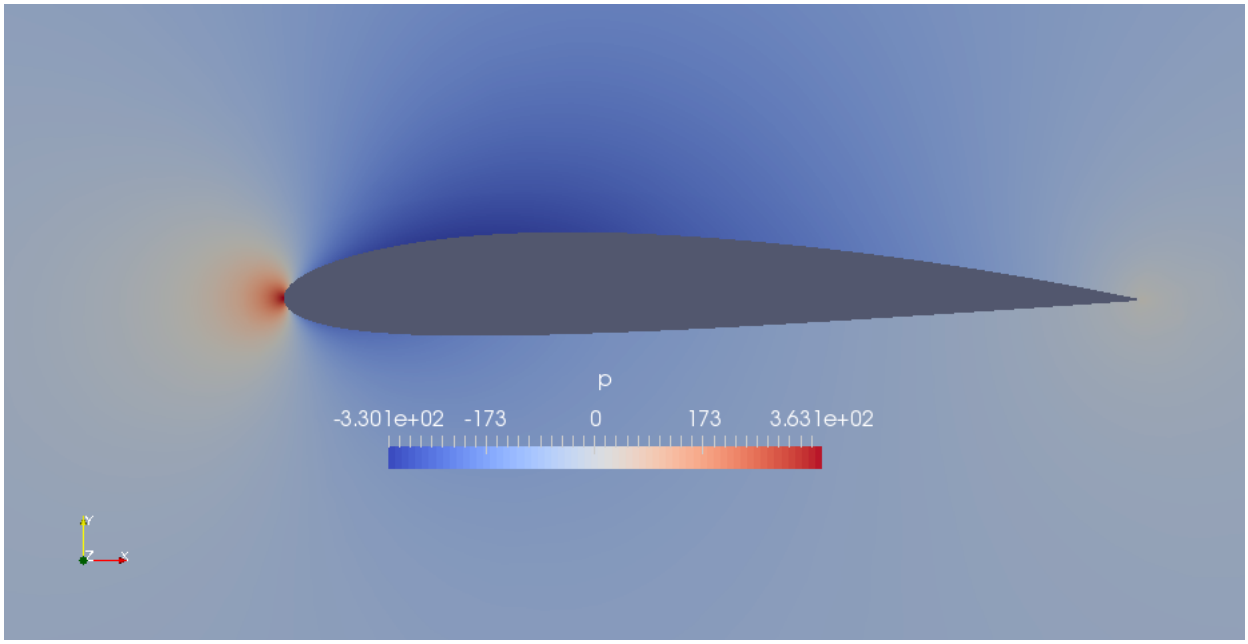
Pro daný úhel náběhu byly vypočteny tyto průběhy veličin U, p a ν_t.



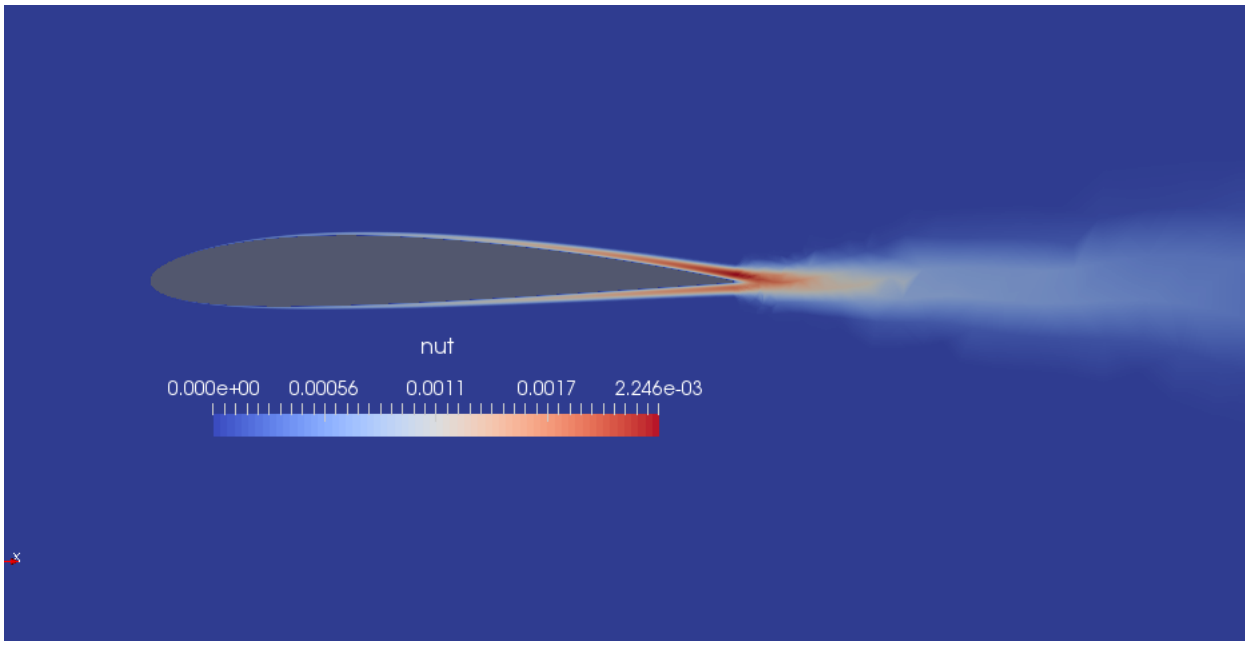
Obrázek 28: Průběh rychlosti v okolí profilu NACA2412 při úhlu náběhu 0^o



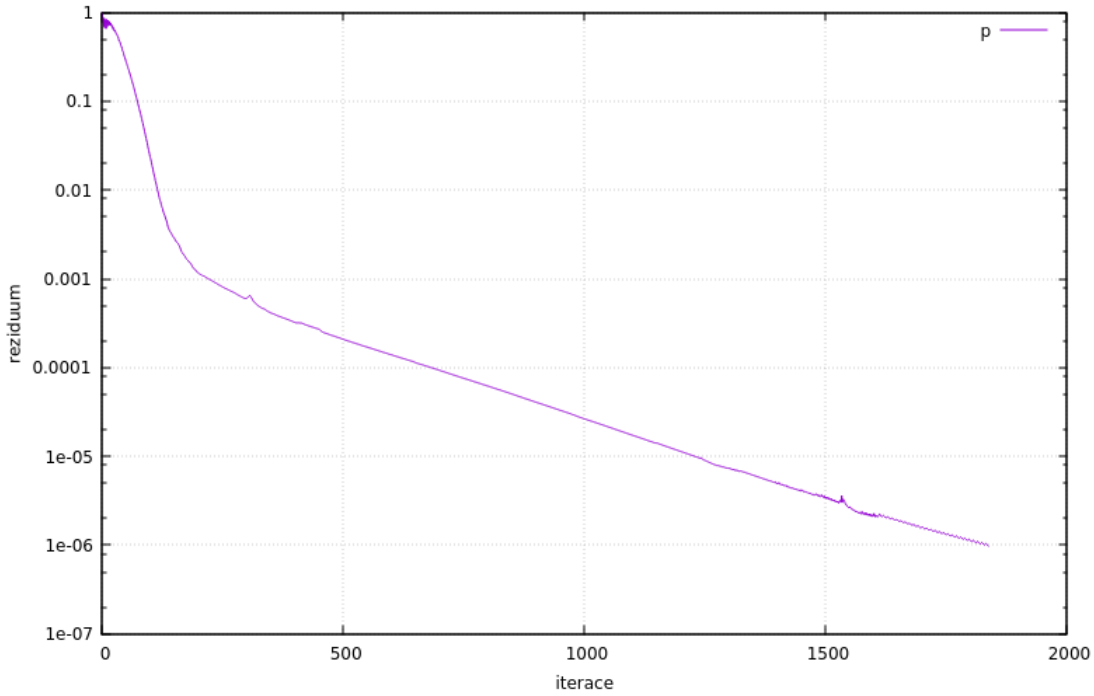
Obrázek 29: Vektorové pole rychlosti v mezní vrstvě



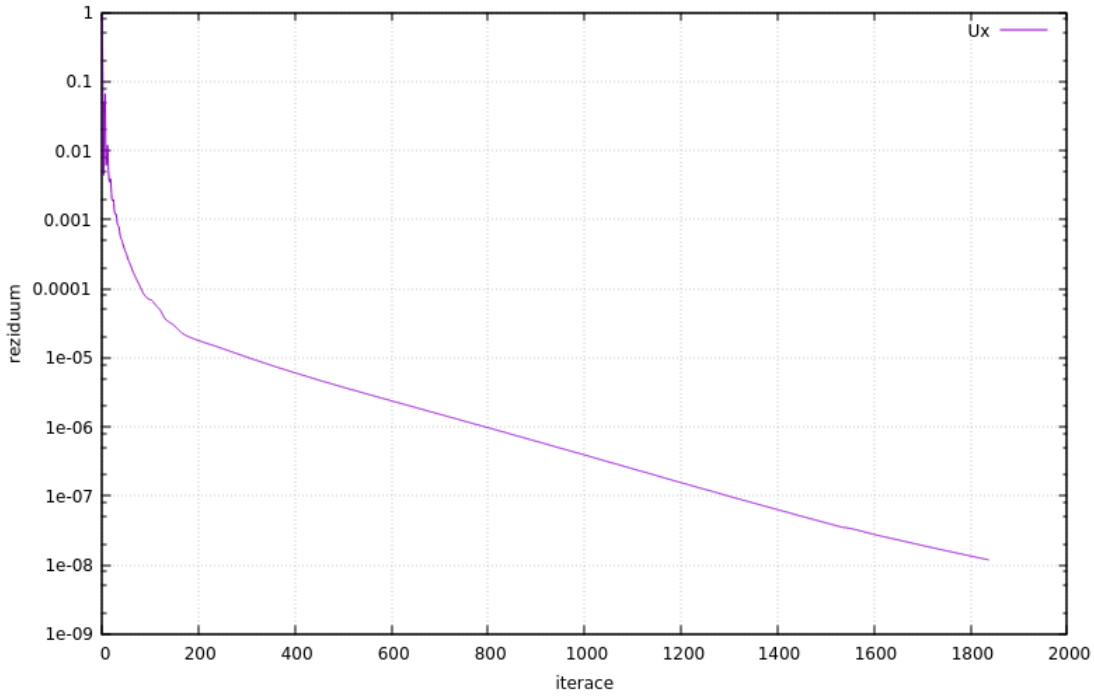
Obrázek 30: Průběh tlaku v okolí profilu NACA2412 při úhlu náběhu 0^o



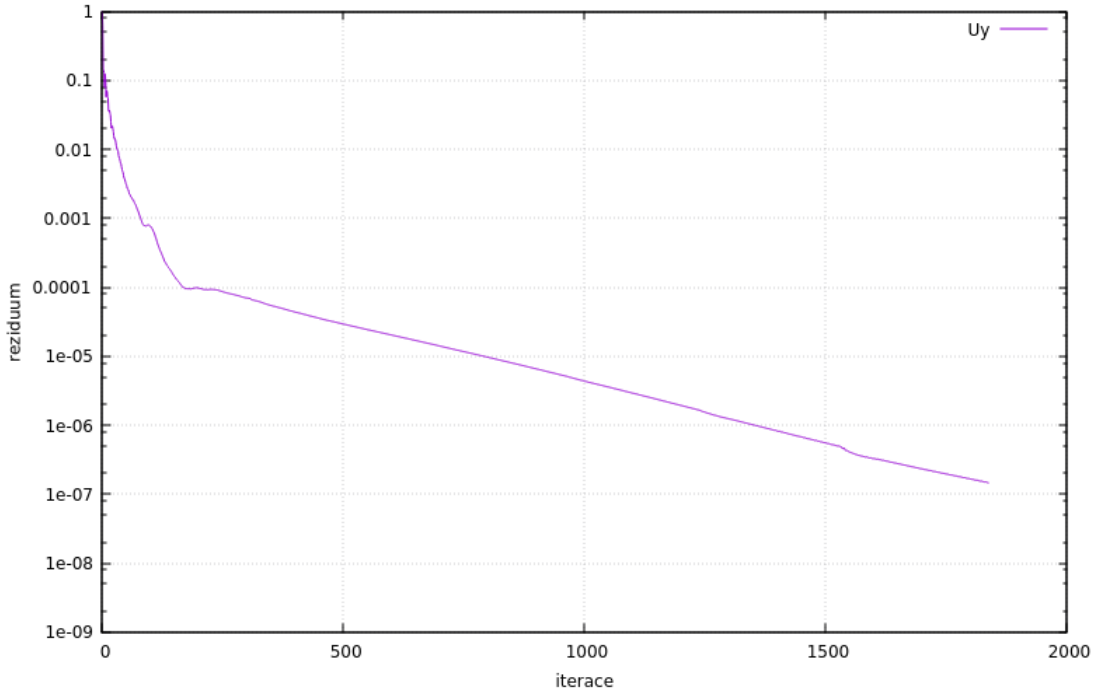
Obrázek 31: Průběh turbulentní viskozity v okolí profilu NACA2412 při úhlu náběhu 0^o



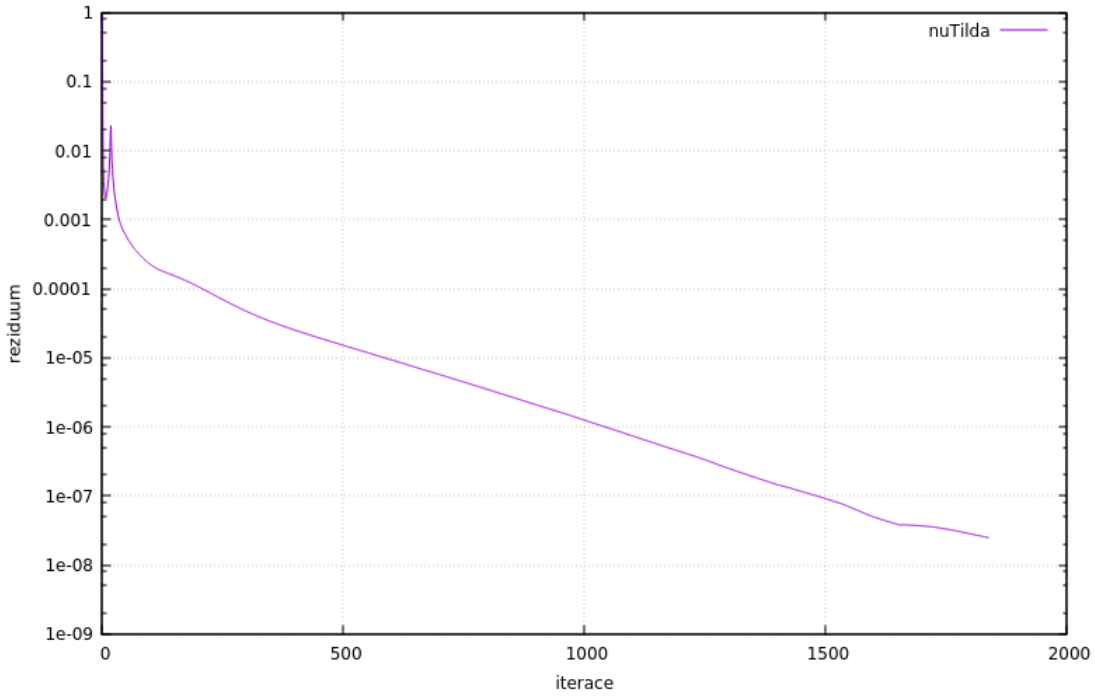
Obrázek 32: Vývoj rezidua tlaku v závislosti na počtu iterací



Obrázek 33: Vývoj rezidua rychlosti ve směru x v závislosti na počtu iterací



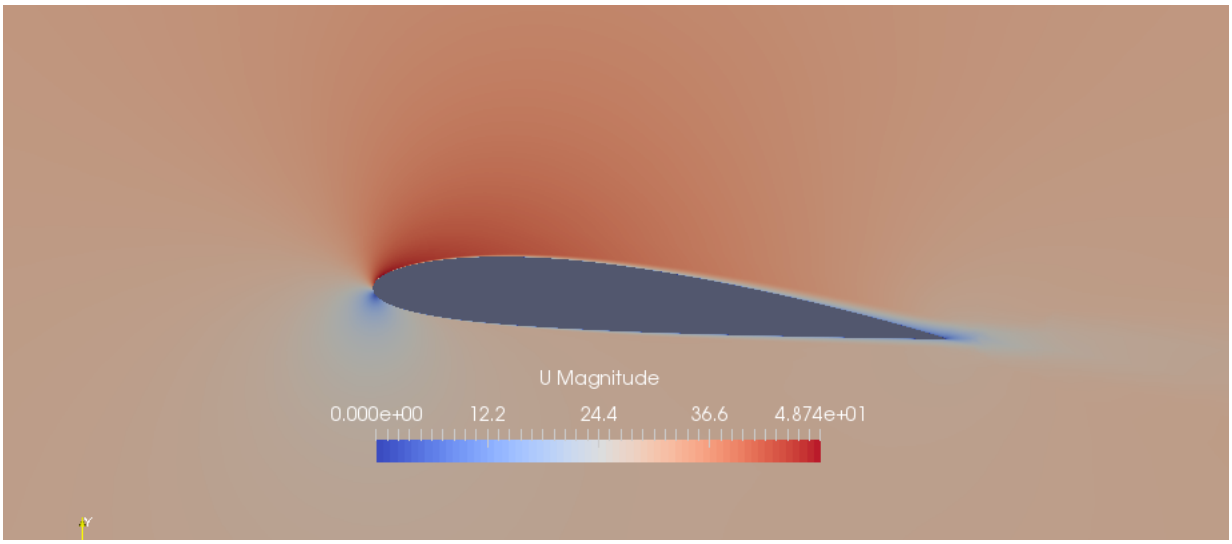
Obrázek 34: Vývoj rezidua rychlosti ve směru y v závislosti na počtu iterací



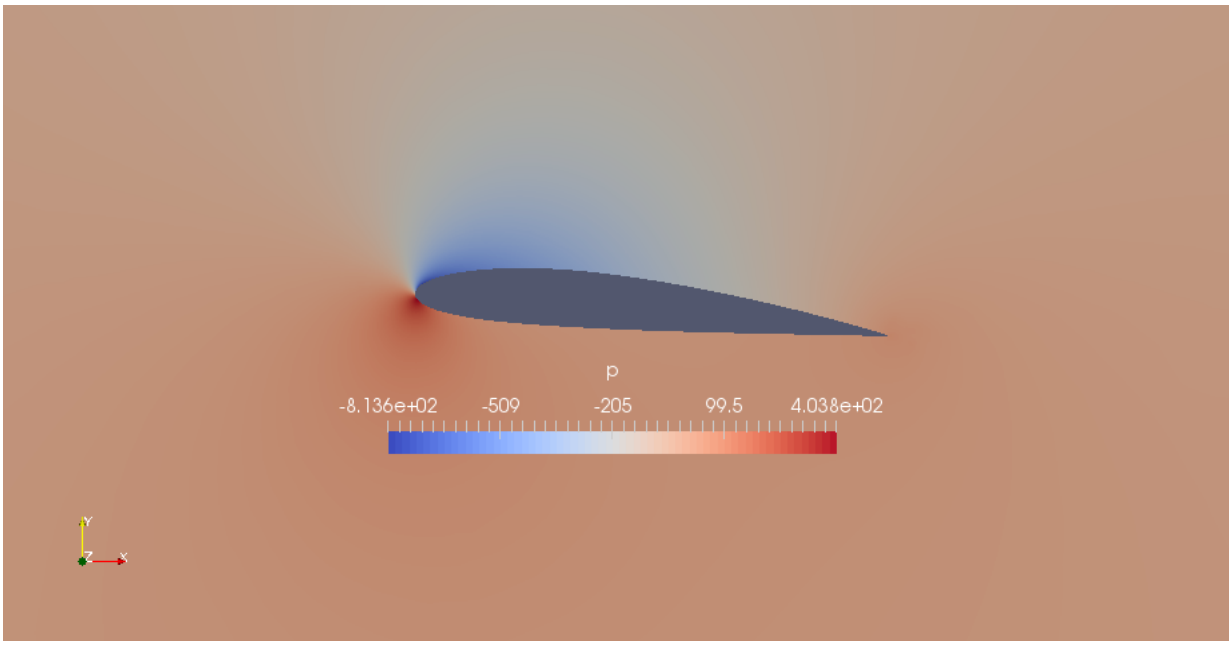
Obrázek 35: Vývoj rezidua $\tilde{\nu}$ v závislosti na počtu iterací

6.1.2 Úhel náběhu 5°

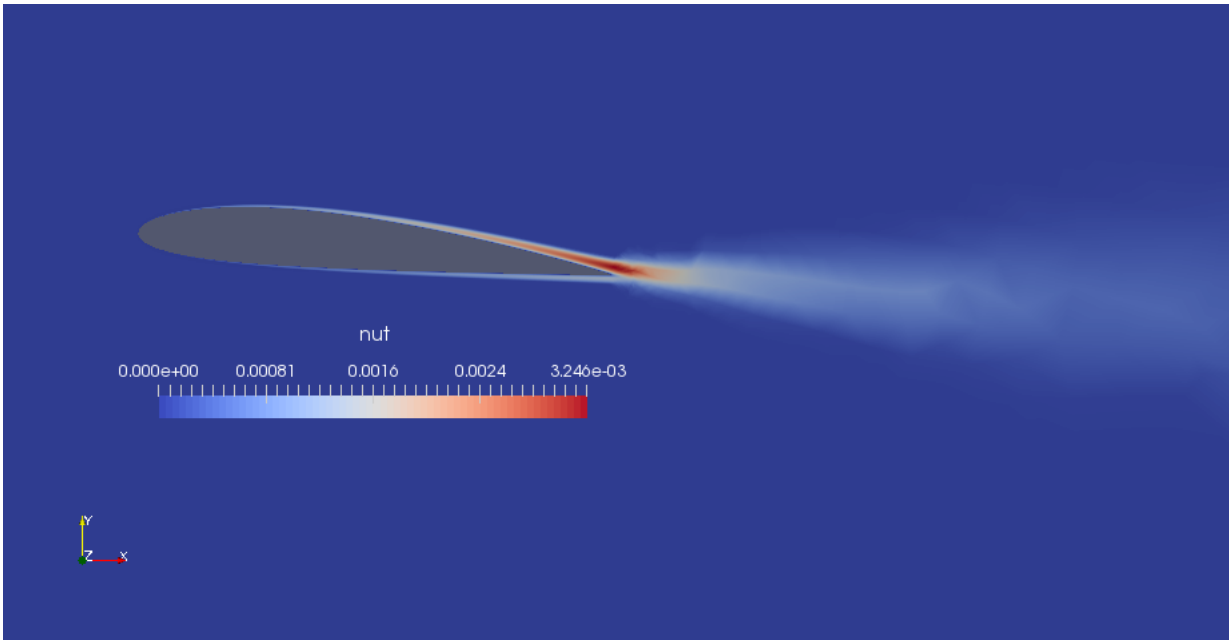
Pro daný úhel náběhu byly vypočteny tyto průběhy veličin U, p a ν_t.



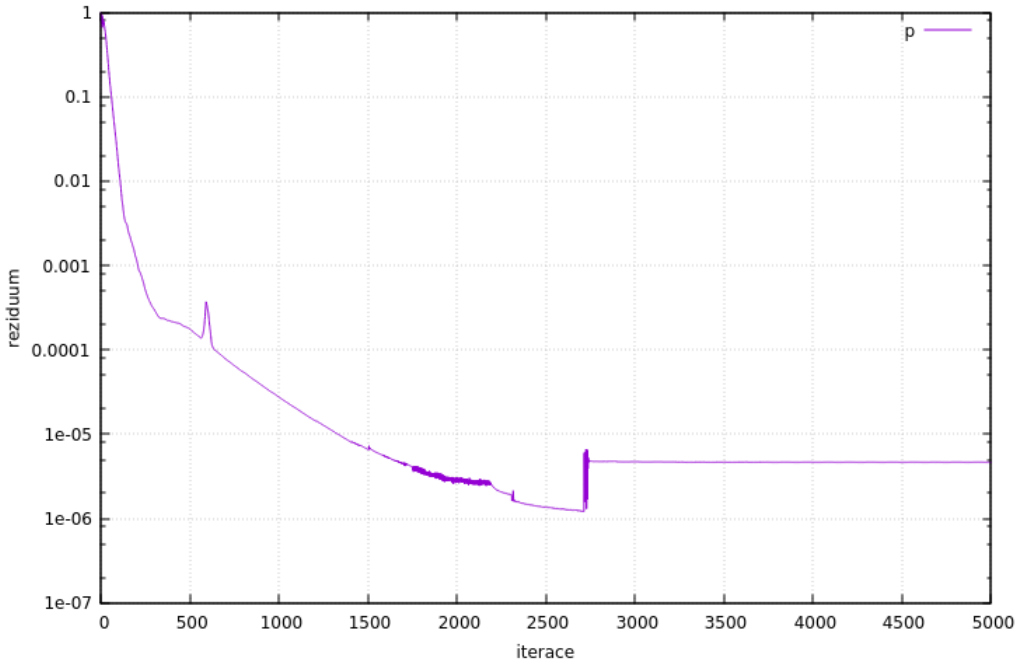
Obrázek 36: Průběh rychlosti v okolí profilu NACA2412 při úhlu náběhu 5^o



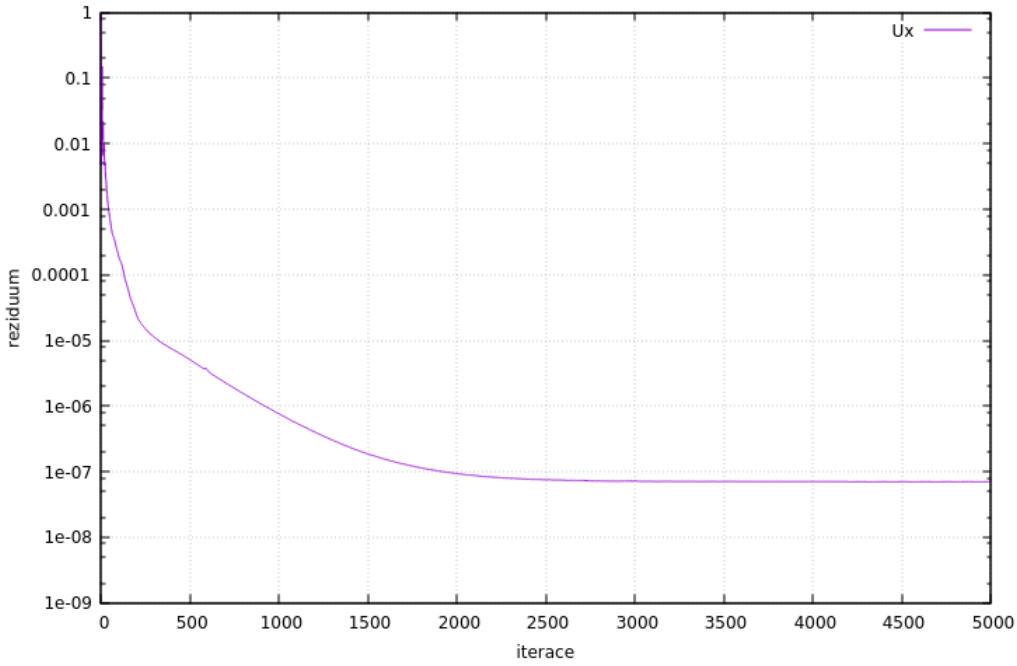
Obrázek 37: Průběh tlaku v okolí profilu NACA2412 při úhlu náběhu 5^o



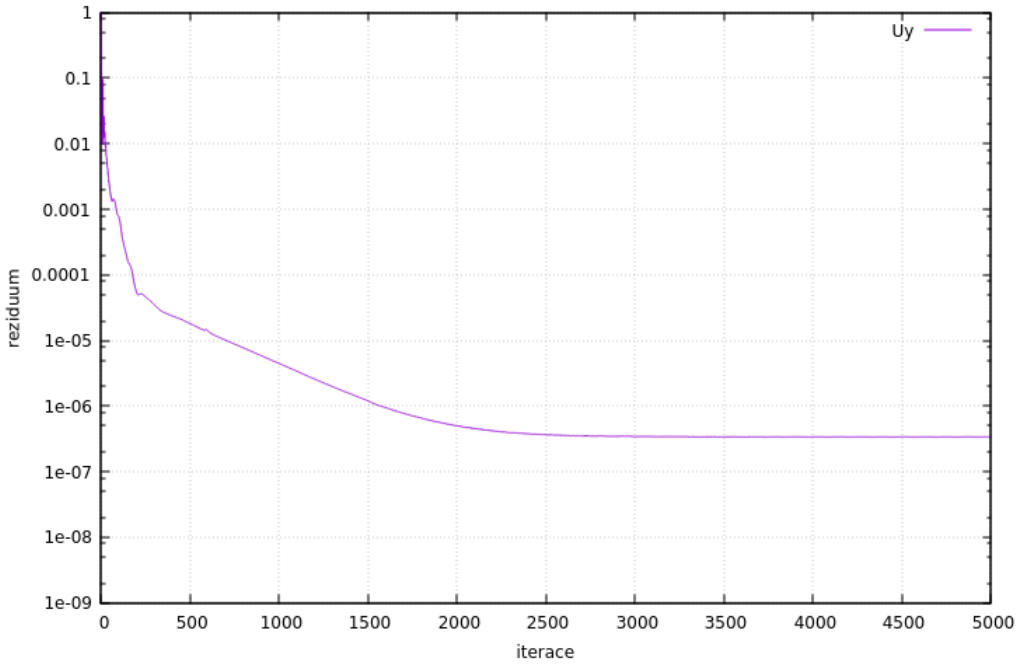
Obrázek 38: Průběh turbulentní viskozity v okolí profilu NACA2412 při úhlu náběhu 5^o



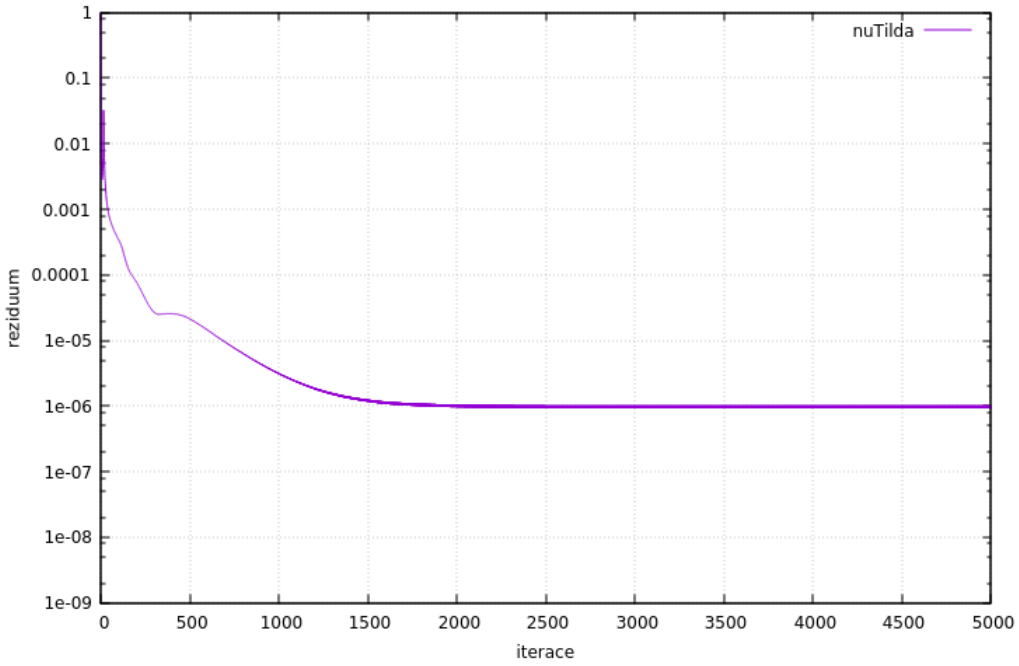
Obrázek 39: Vývoj rezidua tlaku v závislosti na počtu iterací



Obrázek 40: Vývoj rezidua rychlosti ve směru x v závislosti na počtu iterací



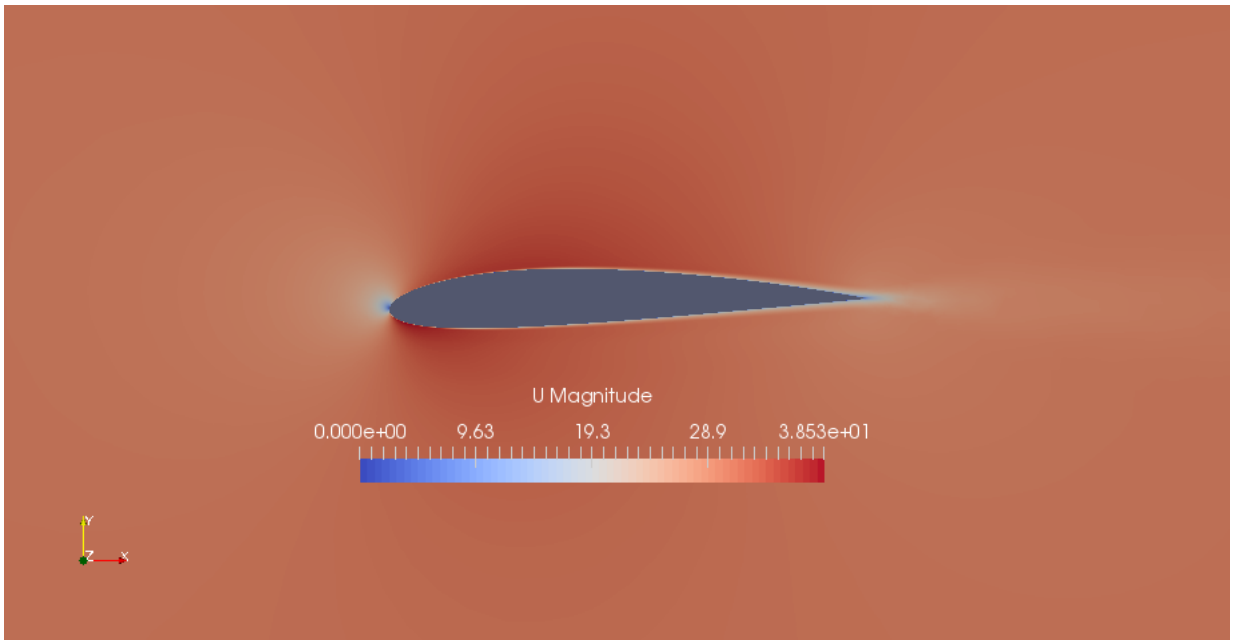
Obrázek 41: Vývoj rezidua rychlosti ve směru y v závislosti na počtu iterací



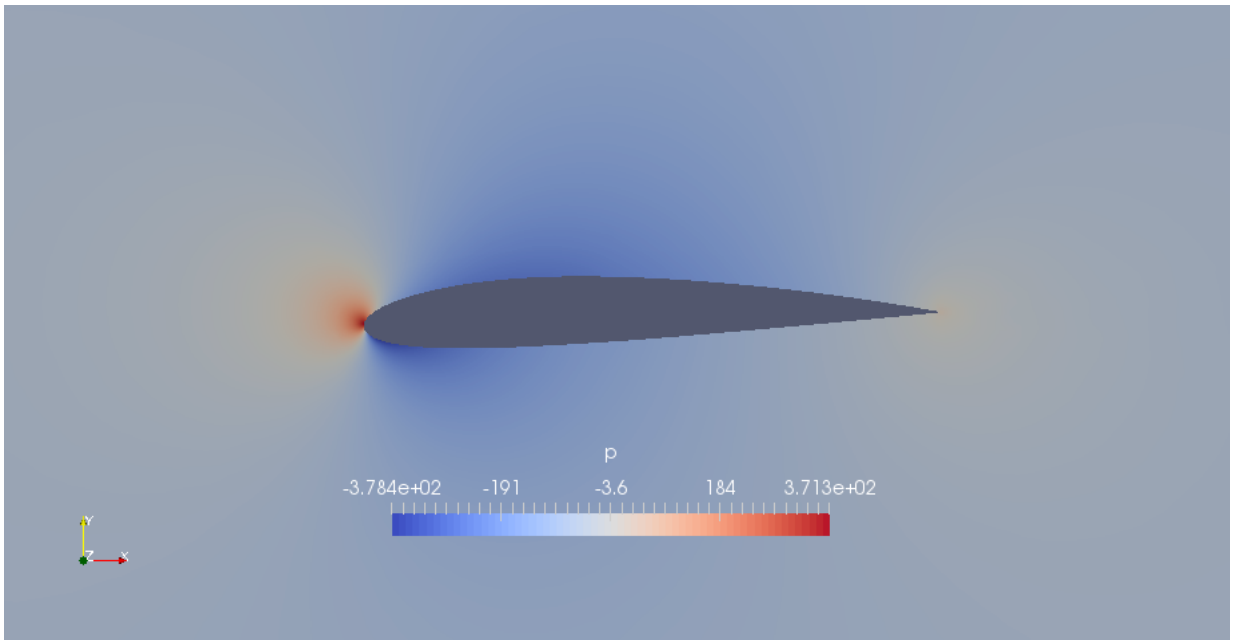
Obrázek 42: Vývoj rezidua $\tilde{\nu}$ v závislosti na počtu iterací

6.1.3 Úhel náběhu -1.5^o

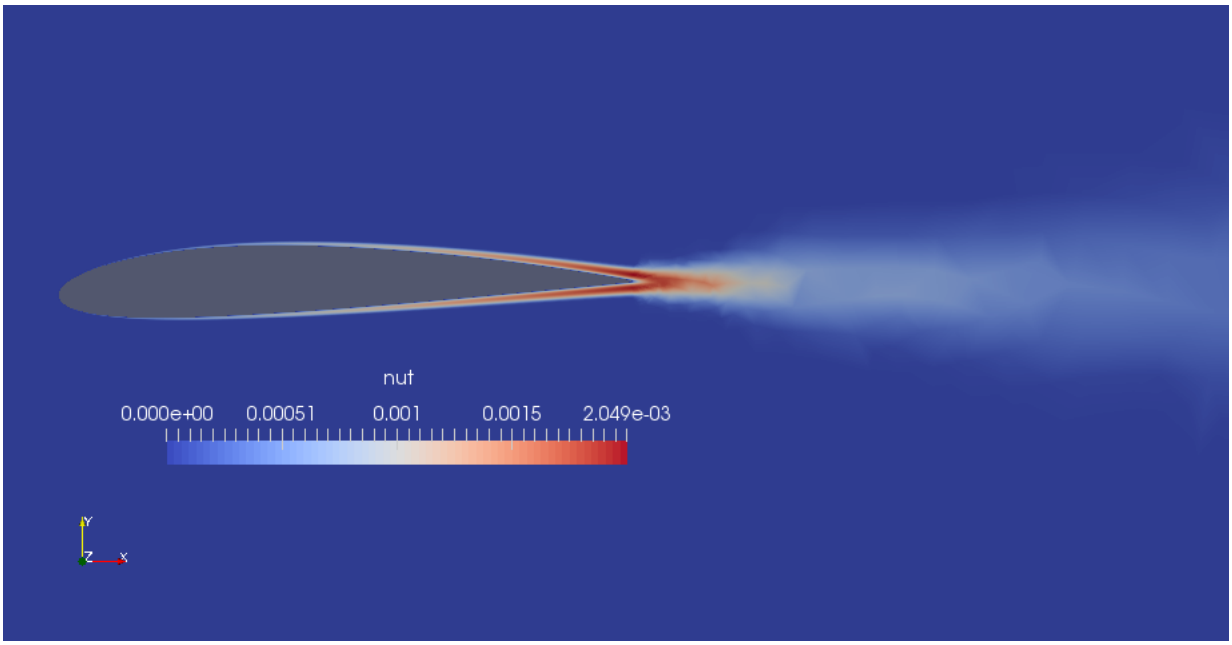
Pro daný úhel náběhu byly vypočteny tyto průběhy veličin U, p a ν_t.



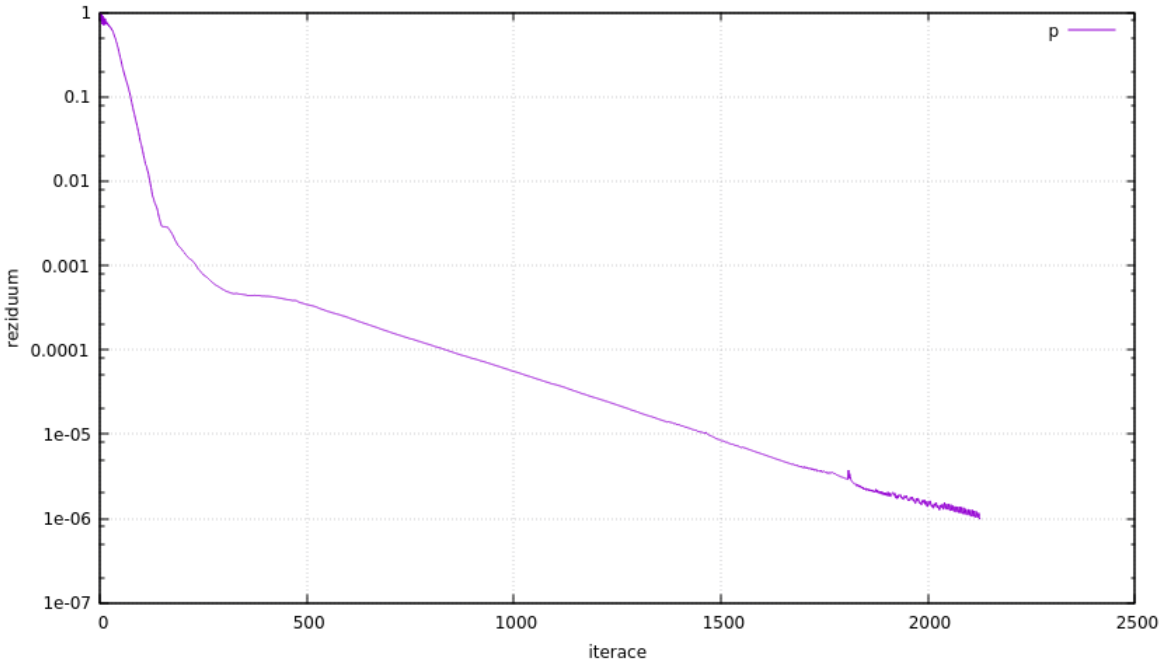
Obrázek 43: Průběh rychlosti v okolí profilu NACA2412 při úhlu náběhu -1.5^o



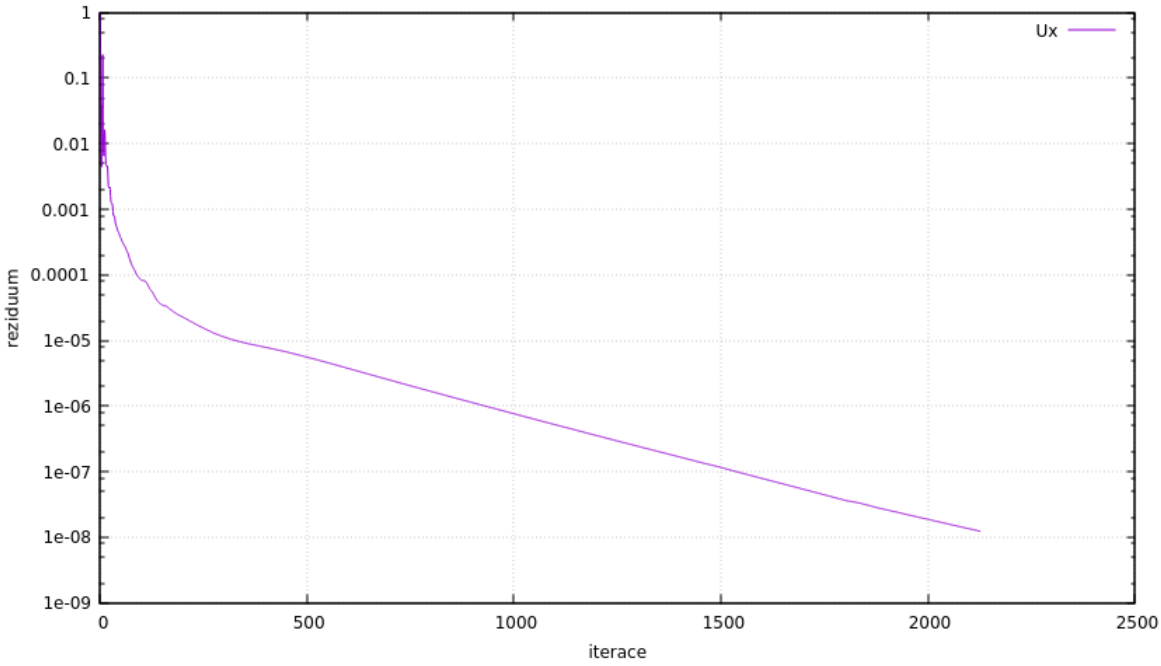
Obrázek 44: Průběh tlaku v okolí profilu NACA2412 při úhlu náběhu -1.5^o



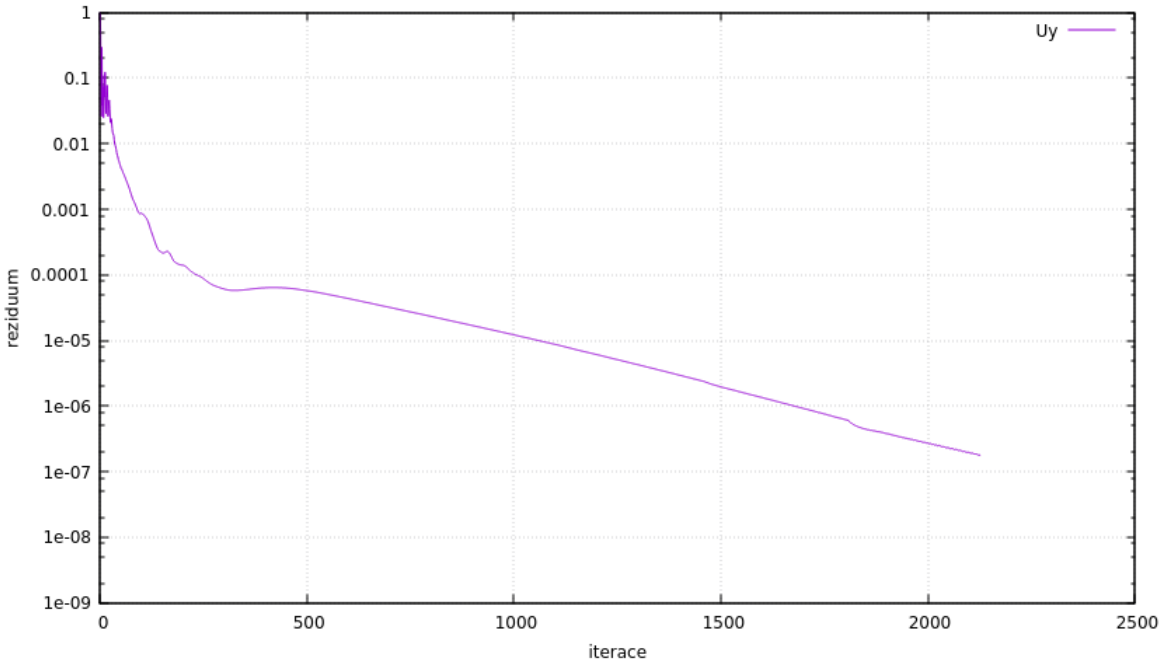
Obrázek 45: Průběh turbulentní viskozity v okolí profilu NACA2412 při úhlu náběhu -1.5^o



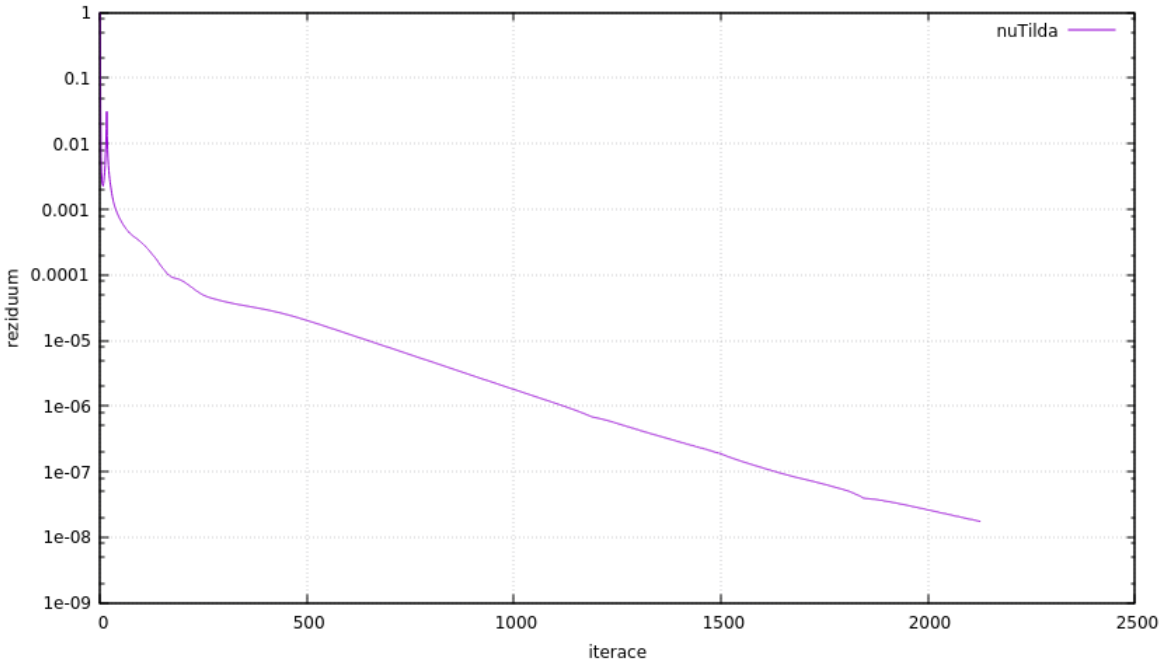
Obrázek 46: Vývoj rezidua tlaku v závislosti na počtu iterací



Obrázek 47: Vývoj rezidua rychlosti ve směru x v závislosti na počtu iterací



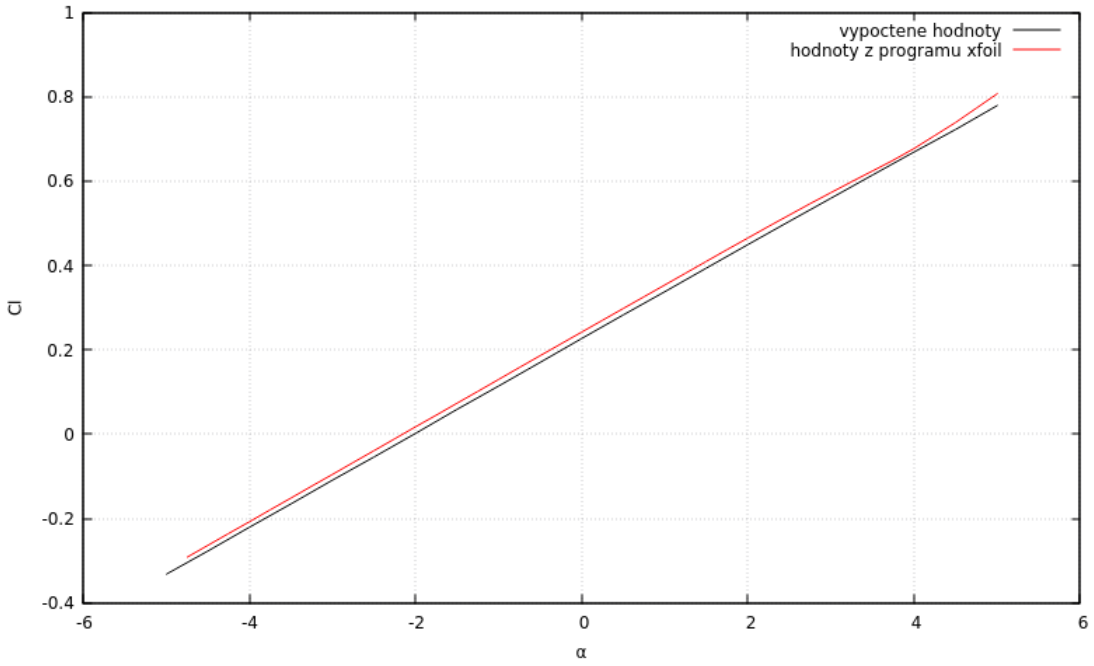
Obrázek 48: Vývoj rezidua rychlosti ve směru y v závislosti na počtu iterací



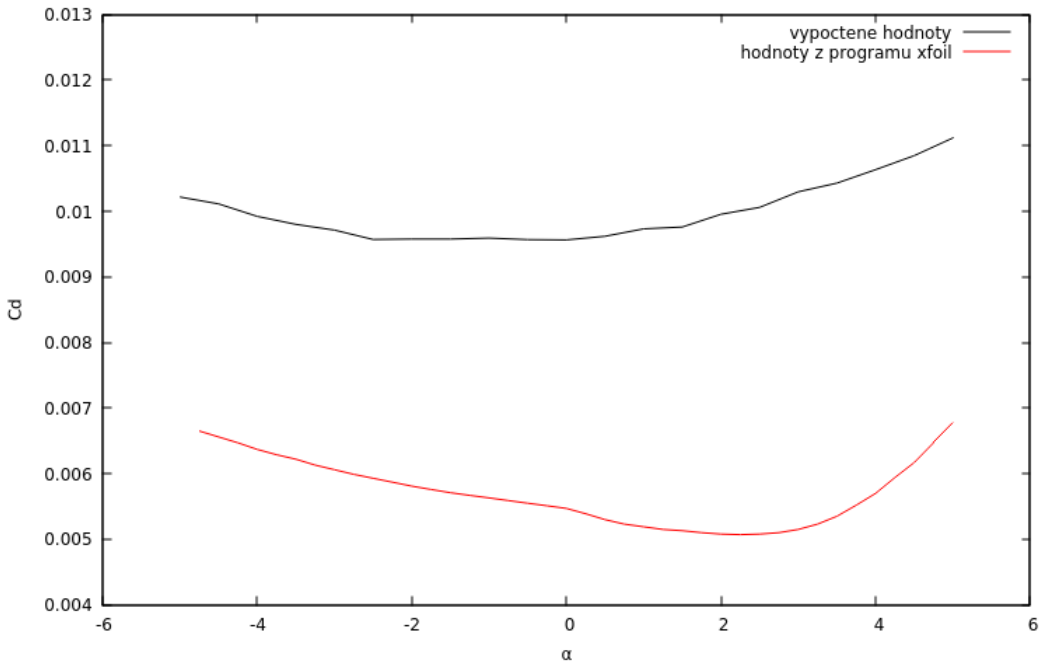
Obrázek 49: Vývoj rezidua $\tilde{\nu}$ v závislosti na počtu iterací

6.1.4 Koeficienty vztlaku a odporu

Ze všech provedených výpočtů, kterých bylo celkem 21, jsem vytvořil závislost koeficientů vztlaku (Cl) a odporu (Cd) na úhlu náběhu α. Pro porovnání jsem do grafů umístil i výsledky získané z programu xFoil, který slouží pro výpočet parametrů profilů křídel.



Obrázek 50: Závislost koeficientu vztlaku na úhlu náběhu pro profil NACA2412 a $Re = 3 \cdot 10^6$



Obrázek 51: Závislost koeficientu odporu na úhlu náběhu pro profil NACA2412 a $Re = 3 \cdot 10^6$

6.2 Vyhodnocení výsledků

V kapitole 6.1.1 jsou zobrazeny výsledky pro úhel náběhu 0^o. Na obrázku (29) vidíme vektorové pole rychlosti v mezní vrstvě stěny profilu. Ten byl vypočten bez použití stěnových funkcí a odpovídá fyzikální představě. Vidíme tedy, že vytvořená síť byla dostatečně jemná. Na obrázcích (28) a (30) je zobrazen průběh velikosti rychlosti a tlaku v okolí profilu. Díky asymetrii profilu NACA2412 nejsou hodnoty těchto veličin stejné pod a nad profilem, který tím pádem generuje vztlak i při nulovém úhlu náběhu. Také je z těchto obrázků zřejmá nepřímá úměrnost mezi tlakem a rychlostí. Obrázek (31) znázorňuje průběh turbulentní viskozity ν_t. Ta není nulová jen v blízkosti profilu křídla, a postupně roste směrem k odtokové hraně profilu. Tam je největší, a za ní se tvoří úplav turbulentního proudění, kde ν_t postupně klesá až na hodnotu neovlivněného proudění. Na obrázcích (32-35) jsou grafy vývoje reziduí jednotlivých veličin s počtem iterací. Vidíme zde, že výpočet dosáhl požadované přesnosti uvedené v kap. 5.3 při zhruba 1800 iteracích.

Pro úhly náběhu 5^o a -1.5^o jsou v kapitolách 6.1.2 a 6.1.3 zobrazeny grafické výsledky stejného významu, který byl popsán v předešlém odstavci. Na obrázcích (39-42) a (46-49) vidíme, že úhel náběhu ovlivňuje konvergenci výpočtu. S rostoucí absolutní hodnotou úhlu náběhu se konvergence výpočtu zhoršuje. Pro úhel -1.5^o výpočet zkonvergoval na požadovanou přesnost po přibližně 2100 iteracích a pro úhel 5^o se přesnost výpočtu zastavila na přibližně o řád méně přesném výsledku, a dále již s přibývajícím počtem iterací neklesala. To je pravděpodobně způsobeno tím, že pro větší úhly náběhu se mění rychlost a tlak na náběžné a odtokové hraně rychleji. Lepší konvergence by se dalo možná dosáhnout zjemněním sítě v těchto místech, ale tím by zároveň vzrostl počet neznámých a zvětšila by se chyba způsobená zaokrouhlením. Takové zjemnění by tedy mohlo nakonec přesnost výpočtu zhoršit. Možných řešení tohoto problému by mohlo být několik: nastavení jiných numerických schémat pro výpočet, změna nastavení přesnosti a relaxačních faktorů nebo použití jiného modelu turbulence. Avšak na tyto změny neexistuje žádný obecný návod a mohly by vést k nestabilitě výpočtu. Z těchto důvodů jsem se spokojil s dosaženou přesností výpočtů, která je pro potřeby této práce dostatečná.

Na obrázku (50) vidíme závislost koeficientu vztlaku c_l na úhlu náběhu $\alpha \in \langle -5; 5 \rangle^o$ získanou ze všech 21 provedených výpočtů. Výsledky odpovídají očekávání a jsou téměř shodné s výsledky získanými z programu xFoil, které jsou zobrazeny v témže grafu. Kvůli již zmíněné asymetrii profilu NACA2412 není vztlak nulový pro $\alpha = 0^o$, ale až pro úhel zhruba -2^o. Na dalším obrázku (51) je závislost koeficientu odporu na úhlu náběhu α. Ta odpovídá hrubé fyzikální představě (odpor roste se zvětšující se absolutní hodnotou úhlu náběhu), avšak hodnoty c_d jsou přibližně dvojnásobné v porovnání s hodnotami z programu xFoil. To je pravděpodobně způsobeno tím, že OpenFOAM počítá s turbulentním prouděním po celé délce profilu. V reálném případě je však na začátku profilu proudění laminární, které se až poté mění na turbulentní. Program xFoil to ve svém výpočtu zohledňuje. Průchod tělesa turbulentním prouděním je kvůli vírům, které obsahuje, náročnější, a proto výsledky z programu OpenFOAM ukazují větší koeficient odporu.

7 ZÁVĚR

V první části této práce byly stručně popsány základní rovnice proudění tekutin a principy jejich numerického řešení. Druhá část byla věnována podrobnému popisu tvorby sítě pomocí programu Gmsh a tvorbě programu k automatickému generování sítě. V další kapitole bylo popsáno nastavení a provedení samotného výpočtu pomocí bezplatného softwarového balíku OpenFOAM. Poslední část byla věnována zobrazení a vyhodnocení získaných výsledků. To bylo provedeno pomocí programů ParaView a Gnuplot.

Výsledky odpovídají fyzikální představě a jsou ve shodě s jinými simulacemi, které můžeme najít například v [4]. Cíle práce byly splněny.

Reference

[1] *OpenFOAM User Guide*. Dostupné z: http://cfd.direct/openfoam/user-guide/, [cit. 2017-06-24].

[2] Kozubková, M. a Šťáva, P., and Drábková, S. *Matematické modely nestlačitelného a stlačitelného proudění: metoda konečných objemů*. Vysoká škola báňská - Technická univerzita, 1999.

[3] Linkeová, I. *Základy počítačového modelování křivek a ploch*. České vysoké učení technické, Fakulta Strojní, 2008.

[4] Saxena, E. S., and Kumar, M. R. Design of naca 2412 and its analysis at different angle of attacks, reynolds numbers, and a wind tunnel test. *International Journal of Engineering and Technology 3* (2015), 193–200.

[5] Šesták, J. Rieger, F. *Přenos hybnosti, tepla a hmoty*. ČVUT, Fakulta Strojní, 1998.

[6] Spalart, P., and Allmaras, S. *A One-Equation Turbulence Model for Aerodynamic Flows*. 1994.

[7] Versteeg, H., and Malalasekera, W. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education Limited, 2007.

Seznam obrázků

SEZNAM PŘÍLOH

- CD-ROM obsahující elektronickou verzi této práce