5 Implementação

Esta é a seção onde descreveremos os módulos e funções principais do nosso script (em `Python`) que funciona como interface entre o software de simulação computacional de fluidos OpenFoam (originalmente escrito em `C++`), o software de otimização contínua Algencan (originalmente escrito em `FORTRAN 90`) e você ☺. Para instruções de instalação, consulte o apêndice.

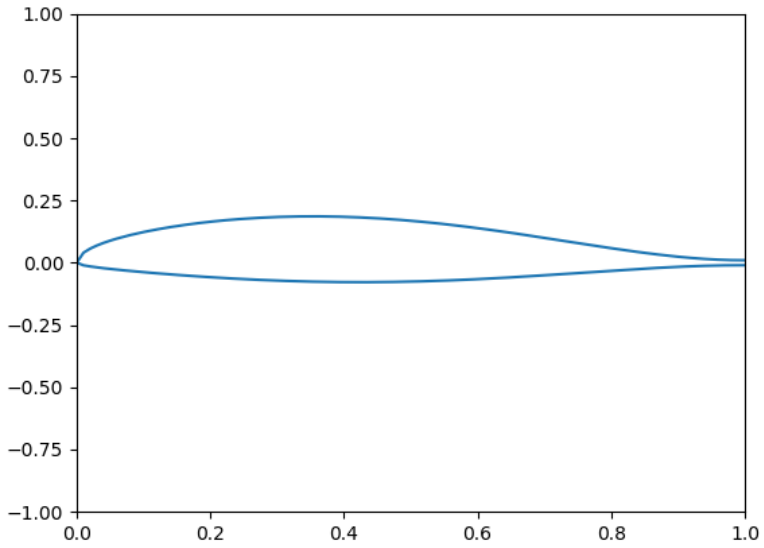
5.1 Descrição do aerofólio via CST

A parametrização via *Class/Shape function Transformation* (CST) é capaz de descrever uma grande classe de aerofólios, incluindo as classes NACA, e chama a atenção por sua simplicidade (Kulfan, 2012). A ideia gira em torno da função $\phi(x) \doteq \sqrt{x}(1-x)$, que descreve uma Borda de Ataque arredondada e uma Borda de Fuga reta e decrescente, já que ela é positiva, possui raízes em $x = 0$ e $x = 1$ e sua derivada tende ao infinito conforme $x \to 0$. Com isto em mente, a parametrização CST nada mais é do que uma generalização de ϕ da forma:

$$\varphi_d(x, N_1, N_2, A, \zeta) = x^{N_1}(1-x)^{N_2}\left[\sum_{r=0}^{d} A_r \binom{d}{r} x^r (1-x)^{d-r}\right] + \zeta x$$

em que $N_1, N_2 \in [0,1]$, $A = [A_0, \ldots, A_d] \in \mathbb{R}^{d+1}$, e 2ζ é o *gap* traseiro entre o Intradorso e o Extradorso. Perceba que $\varphi_0(x, 0.5, 1, 0, 0) = \phi(x)$. Os *coeficientes de forma* (armazenados no vetor A) que descrevem o Intradorso serão chamados de A_l, e os que descrevem o Extradorso, de A_u. Usaremos os mesmos valores de N_1 e N_2 para ambos, Intra- e Extradorso, e fixaremos $\zeta = 0$ por conveniência. Além disso, perceba que o fator entre colchetes na expressão de φ nada mais é do que um polinômio arbitrário de grau d escrito na base de Bernstein; sendo assim d será um parâmetro fixo, e não uma variável.

Por exemplo, com $d = 2$, $N_1 = 0.5$, $N_2 = 1$, $\zeta = 0.01$, e $A_l = [0.1, 0.2, 0.3]$ e $A_u = [0.4, 0.5, 0.6]$, temos:



Em nosso script, a função que constrói esta curva é chamada

```
cst(Au,Al,par)
```

que se encontra no módulo `main.py`, onde `Au` e `Al` são os vetores de coeficientes A_u e A_l, respectivamente, e `par` é um dicionário da forma:

```
par={
    'n':          número de pontos gerados,
    'deg':        grau do polinômio,
    'N1':         N1,
    'N2':         N2,
    'trail_gap':  zeta
}
```

5.2 Discretização do domínio

O sistema de equações diferenciais que modelam o ar (descrito nas seções anteriores) é resolvido numericamente pelo OpenFoam com base nos parâmetros que o usuário fornece. Inclusive, é válido mencionar que ele oferece bastante liberdade na escolha dos parâmetros. Um dos principais componentes desses parâmetros é a *malha de discretização* do domínio das variáveis U e p (e, caso haja turbulência, de ν_t e $\tilde{\nu}$ também).

O desenho de malha mais comum para o estudo de aerofólios consiste numa caixa formada pela união entre um círculo e um retângulo, menos o aerofólio, que estaria no centro do círculo (veja as Figuras 18 e 19).

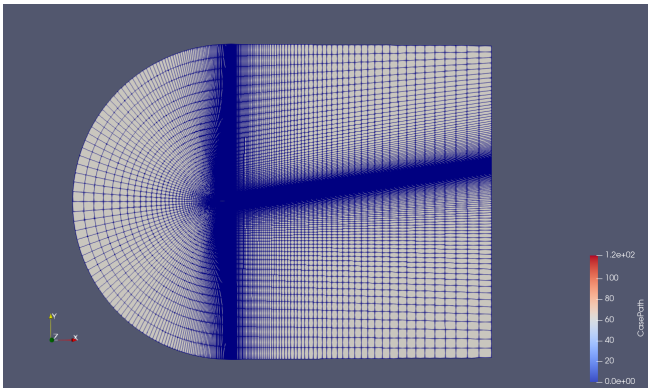


Figura 18: Uma malha vista de longe.

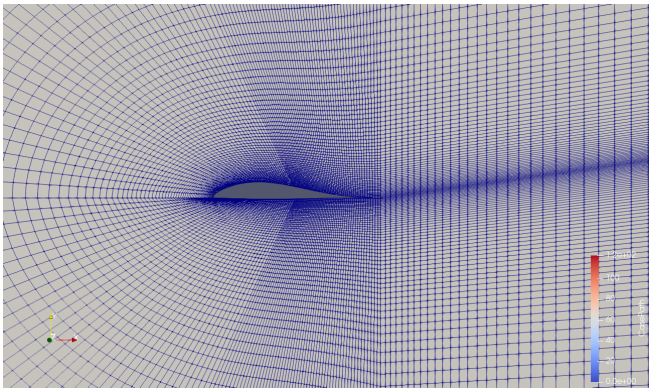


Figura 19: Zoom em torno do aerofólio.

O próprio OpenFoam possui uma ferramenta para a construção de malhas, chamada `blockMesh`, que se alimenta de um dicionário chamado `blockMeshDict` na pasta `system`. Tenha em mente que construir boas malhas é uma arte e que o `blockMesh` não faz tudo sozinho. Para poupar o seu tempo, nós preparamos um script básico que escreve um arquivo `blockMeshDict` automaticamente, e pode ser usado para gerar o desenho simples de malha que descrevemos anteriormente, chamado

```
write_blockMeshDict(x,y,param)
```

que se encontra no módulo `writer.py`. Os argumentos `x` e `y` são as coordenadas $(x, \varphi_d(x, \cdot))$ do aerofólio, onde $x \in [0, 1]$. Não é preciso se preocupar com as coordenadas, já que elas são dadas pela função `cst`. Foquemos em `param`, que é um dicionário que descreve alguns aspectos fundamentais da malha, dado por:

```
param={
    'n':                    número de pontos do contorno do aerofólio,
    'scale':                fator de escala (padrão: 1),
    'radius':               distância entre a Borda de Ataque e o primeiro
                                ponto de entrada de ar (raio do círculo),
    'dist_x_out':           distância entre a Borda de Fuga e o último ponto
                                de saída de ar (comprimento do retângulo),
    'depth':                profundidade (padrão: 0.3),
    'angle':                ângulo de ataque,
    'exp_ratio':            razão de expansão > 1 (maior significa mais
                                espaçamento longe do aerofólio),
    'bound_thick':          determina a densidade da malha no contorno do
                                aerofólio > 0 (menor significa mais denso),
    'first_thick':          densidade da primeira camada da malha (menor
                                significa mais denso),
    'max_cell_size_in':     nível de refinamento na fronteira externa
                                (inlet, padrão: 1),
    'max_cell_size_out':    nível de refinamento na fronteira externa
                                (outlet, padrão: 1),
    'max_cell_size_in_out': nível de refinamento na fronteira externa
                                (inlet x outlet, padrão: 1),
    'cell_size_middle':     *comprimento da célula fora das Bordas de Ataque
                                e Fuga,
    'cell_size_trail':      *comprimento da célula na Borda de Fuga,
    'cell_size_lead':       *comprimento da célula na Borda de Ataque,
    'sep_point':            ponto de separação entre 0.1 e 0.9 (padrão: 0.5)
}
```

Os parâmetros principais da malha são `cell_size_middle/trail/lead`, destacados com `*`, pois são eles que determinam o nível de refinamento em torno do aerofólio. Antes de executar qualquer teste, nós recomendamos ajustar estes parâmetros (empiricamente ou não) de modo a obter `Mesh OK` como saída do `checkMesh`. Nós recomendamos consultar a documentação do OpenFoam na Wiki oficial para aprender a interpretar o `checkMesh`. Para ilustrar os efeitos dos parâmetros principais, veja as Figuras 20 e 21. Repare que a malha da Figura 21 é grotescamente mais fina que a da Figura 20 com respeito ao tamanho das células em torno do aerofólio, mas ambas estão `OK` para o `checkMesh`. Os parâmetros principais estão especificados nas legendas as figuras, enquanto os outros parâmetros coincidem com os valores padrão[1].

[1] **Nota dos autores:** vale a pena investir algum tempo para mexer nos parâmetros e verificar a malha resultante a cada mudança, a fim de adquirir um pouco de intuição sobre eles.

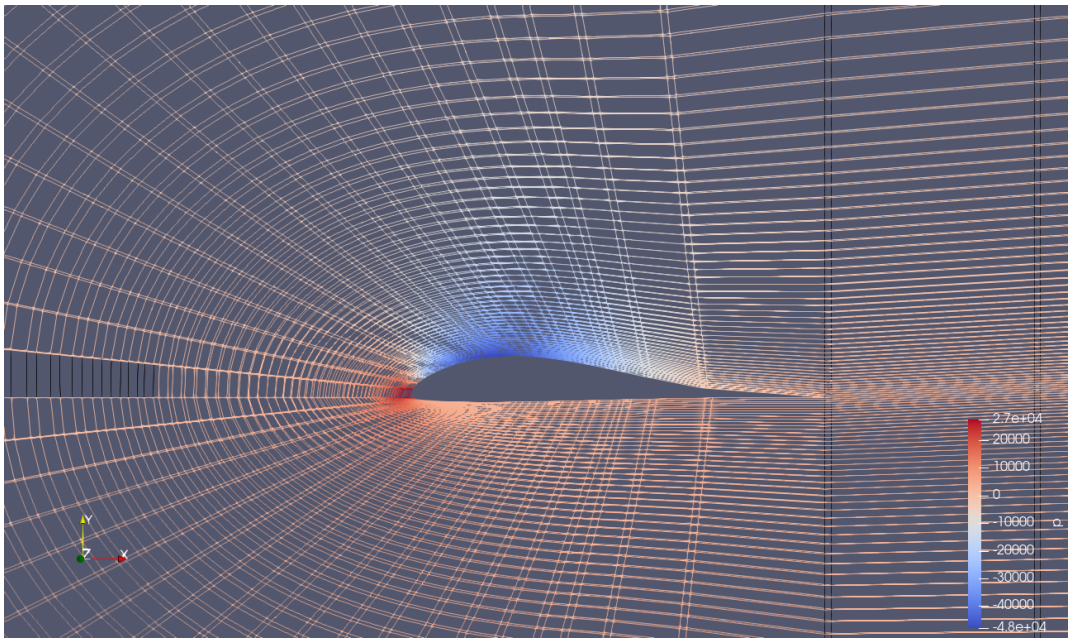


Figura 20: middle= 0.05, trail= 0.2, lead= 0.02.

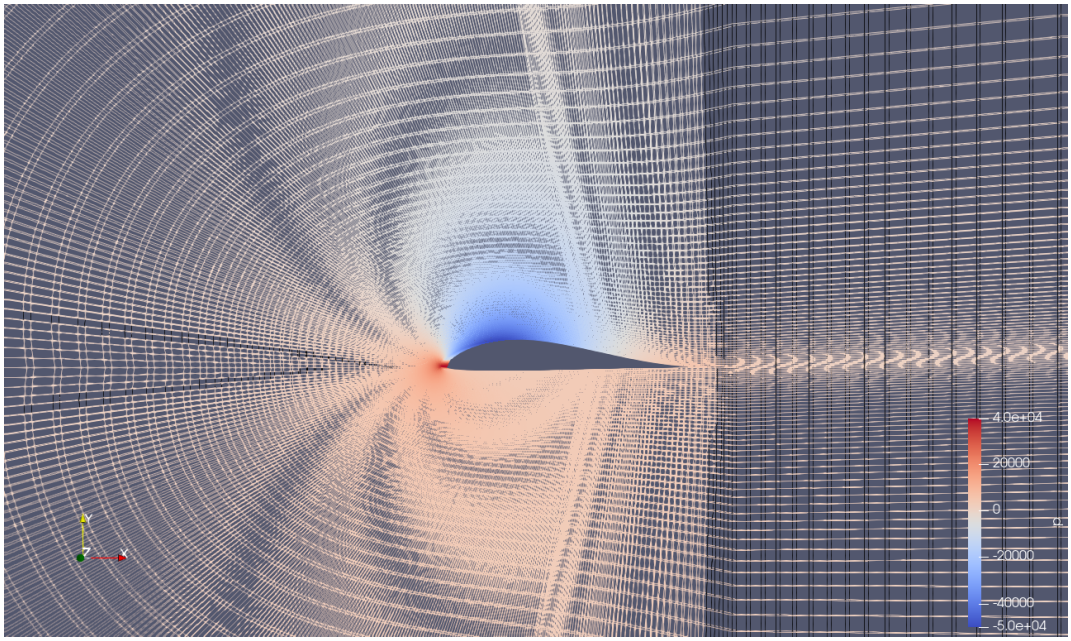


Figura 21: middle= 0.005, trail= 0.02, lead= 0.002.

5.3 Condições de contorno e controle de forças

As condições de contorno do problema que será resolvido numericamente são especificadas na pasta `0` (que corresponde à iteração zero), em dicionários. Há um para cada variável do problema,

que no caso são: a velocidade U, a pressão p e as duas variáveis de turbulência ν_t e $\tilde{\nu}$. Existe um script no módulo `writer.py` chamado

```
write_boundaryCond(boundary)
```

que escreve esses dicionários com base na entrada fornecida pelo usuário contendo os seguintes campos:

```
boundary={
    'u1':        velocidade no eixo 1,
    'u2':        velocidade no eixo 2,
    'u3':        velocidade no eixo 3 (padrão: 0),

    'p':         pressão (padrão: 0),

    'nut':       (padrão: 22% da viscosidade cinemática),
    'nuTilda':   (padrão: 400% da viscosidade cinemática)
}
```

É importante lembrar que as componentes da velocidade em cada eixo são calculadas com base no ângulo de ataque α e que um dos eixos é vazio (no nosso caso, o eixo 3). Por exemplo, se o eixo 3 é vazio e a velocidade do ar em *free stream* é denotada por U_∞, então $u_1 = U_\infty \cos(\alpha)$ e $u_2 = U_\infty \sin(\alpha)$ e, é claro, $u_3 = 0$.

Além disso, os parâmetros básicos de controle do problema são especificados no dicionário `controlDict` na pasta `system`. Novamente, para facilitar a sua vida, existe uma script chamado

```
write_controlDict(control)
```

no módulo `writer.py` que escreve este dicionário com base nos seguintes parâmetros:

```
control={
    't0':         índice da primeira iteração (padrão: 0),
    'tf':         número máximo de iterações (padrão: 20000),
    'dt':         tamanho de passo (irrelevante em regime estacionário),
    'writeint':   intervalo de escrita (padrão: 50),
    'purge':      número de iterações salvas (padrão: 2)
}
```

Para computar as forças de arrasto e sustentação (e consequentemente os coeficientes C_D e C_L) basta pedir isso ao OpenFoam por meio de um dicionário extra em `controlDict` chamado `forceCoeffs`, que é escrito pela função

```
write_forceCoeffs(forces_control)
```

em `writer.py` a partir dos parâmetros:

```
forces_control={
    'AoA':       ângulo de ataque,
    'rhoInf':    densidade do ar (padrão: 1.225),
    'magUInf':   velocidade do fluido em free stream (m/s),
    'lRef':      referência de comprimento (padrão: 1),
    'Aref':      referência de área (padrão: 1),
    'timeint':   intervalo de tempo (irrelevante),
    'log':       imprimir iterações (true) ou não (false)
}
```

Com isto, os coeficientes e forças de arrasto e sustentação serão calculados automaticamente pelo OpenFoam e escritos no arquivo `postProcessing/forceCoeffs/0/coefficient_0.dat` na última linha.

5.4 SIMPLE

SIMPLE é o algoritmo que resolve as equações de Navier-Stokes em sua forma reduzida, chamado pelo comando `simpleFoam`. O usuário tem acesso a algum nível de customização quanto aos métodos empregados para resolver os subproblemas do SIMPLE e seus parâmetros. Este acesso é feito por meio dos dicionários `fvSchemes` e `fvSolution` na pasta `system`. Há um script em `writer.py` chamado

```
write_fvSolution(fvsolution)
```

que escreve `fvSolution` e se alimenta dos parâmetros:

```
fvsolution={
    'ptol':            tolerância absoluta em p (X significa 10e-X),
    'preltol':         tolerância relativa em p (X significa 10e-X),

    'Utol':            tolerância absoluta em U (X significa 10e-X),
    'Ureltol':         tolerância relativa em U (X significa 10e-X),
    'Usweeps':         número de sweeps em U (padrão: 2),

    'nuTildatol':      tolerância absoluta em nu-tilde (X significa 10e-X),
    'nuTildareltol':   tolerância relativa em nu-tilde (X significa 10e-X),
    'nuTildasweeps':   número de sweeps em nu-tilde (padrão: 2),

    'nonortho':        número de correções de não-ortogonalidade (padrão: 2),

    'pres':            critério de parada em p (X significa 10e-X),
    'Ures':            critério de parada em U (X significa 10e-X),
    'nuTildares':      critério de parada em nu-tilde (X significa 10e-X),

    'pfact':           fator de relaxação em p (padrão: 0.3),
```

```
    'Ufact':           fator de relaxação em U (padrão: 0.7),
    'nuTildafact':     fator de relaxação em nu-tilde (padrão: 0.7),
}
```

Por outro lado, não é esperado que o dicionário `fvSchemes` seja alterado durante a execução de outro algoritmo e, portanto, o usuário deve alterá-lo manualmente caso necessário. É muito importante ter em mente que estes dicionários afetam muito o desempenho do método, com destaque para a escolha do *solver* descrito no campo `divSchemes` em `fvSchemes`. Após alguns testes, concluímos que `bounded Gauss upwind` é a melhor opção para computar o coeficiente de arrasto em nosso problema. As outras opções de *solver* dificultaram a convergência do método, em particular, com respeito à variável $\tilde{\nu}$. Nós fixamos os critérios de parada em 10^{-5} para todas as variáveis em nossos testes.

5.5 Interfaces

No módulo `main.py` há um script chamado

```
airfoil_data(Au,Al,par,param,boundary,control,forces_control,fvsolution)
```

cuja função é criar uma malha dado um contorno de aerofólio e alguns parâmetros, executar o `simpleFoam`, coletar os coeficientes de arrasto e sustentação, e apagar o lixo gerado por esta execução. Este script recebe como entrada todos os parâmetros descritos anteriormente, que devem ser fornecidos no módulo `airfoil.py`, e retorna todos os dados de forças do aerofólio fornecido.

Além disso, quem integra o OpenFoam com o algoritmo de otimização Algencan, que será descrito a seguir, é a função

```
feval(n,param,boundary,control,forces_control,fvsolution)
```

também do módulo `main.py` que lê os dados de um aerofólio a partir de um arquivo de entrada `eval/feval.in` que possui o formato `N1 N2 Au Al`, executa `airfoil_data` e escreve os coeficientes de arrasto e sustentação num arquivo de saída `eval/feval.out`.

5.6 Um breve resumo e algumas instruções

Existem três módulos principais em nosso programa:

- `airfoil.py` é onde são fornecidas as condições de contorno do problema e os parâmetros de controle e da malha; e também é o script que chama `feval` com base nessas informações;
- `main.py` é onde estão implementadas as interfaces com o OpenFoam e a construção do contorno do aerofólio via CST;
- `writer.py` é onde se encontram as funções que escrevem os dicionários de entrada para o OpenFoam.

Para calcular os coeficientes de arrasto e sustentação de um dado aerofólio, basta inserir os coeficientes CST no arquivo `feval.in` na pasta `eval` seguindo o seguinte formato:

$$N_1 \quad N_2 \quad A_u \quad A_l$$

em seguida ajuste os parâmetros em `airfoil.py` e execute o script com o comando

```
python airfoil.py
```

num terminal do Linux, a partir da pasta onde o script se encontra. Aguarde a execução do método e verifique os coeficientes de arrasto (C_D) e sustentação (C_L) computados, que estarão escritos no arquivo `feval.out` na pasta `eval`, no seguinte formato:

$$C_D \quad C_L$$

Os dados de saída do `simpleFoam` serão armazenados no arquivo `out_simpleFoam.txt` e as informações sobre a malha geradas pelo `checkMesh` estarão em `out_checkMesh.txt`. Note que as pastas com os valores finais das variáveis computados pelo `simpleFoam` foram excluídas; logo, se você quiser visualizá-las por meio de um software como o ParaView, você deve executar o `simpleFoam` novamente. Vale lembrar que você pode usar a função `paraFoam` do OpenFoam para visualizar os dados com o ParaView.

5.7 Algencan

Para computar um aerofólio ótimo usando o Algencan, basta editar o arquivo `air.f90`, compilá-lo, e executá-lo com

```
gfortran -O3 air.f90 -L$algencan-3.1.1/lib -lalgencan -o air
```

```
./air
```

Contudo, devemos ter em mente que existem mais coisas a serem modificadas para a utilização do Algencan. No programa principal, chamado `algencanma`, você deve começar modificando o número de variáveis `n` e os limitantes inferiores e superiores de cada variável `l` e `u`. O ponto inicial será lido de `eval/feval.in`.

Se, por acaso, você tiver acesso às derivadas das suas restrições ou função objetivo, você deve informar isso ao Algencan por meio do vetor `coded` de valores binários. Além disso, os parâmetros de precisão para a viabilidade e otimalidade são definidos por `epsfeas` e `epsopt`, respectivamente.

Agora você deve modificar as funções do problema:

```
myevalf(n,x,f,flag)
```

```
myevalc(x,n,ind,c,flag)
```

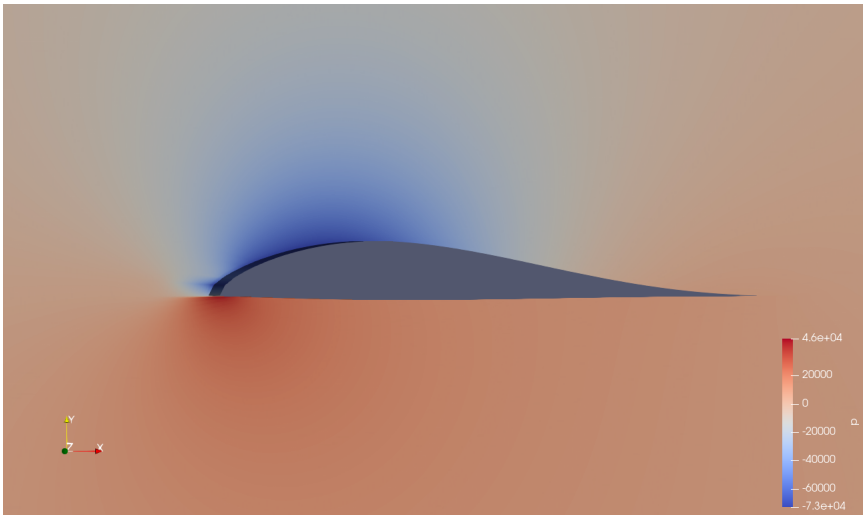
que correspondem à função objetivo e à `ind`-ésima restrição de desigualdade, respectivamente. E, se for o caso, forneça suas derivadas também codificando `myevalg` e `myevaljac`, respectivamente. Em nosso código, note que apenas a função objetivo `myevalf` está codificada e que ela consiste apenas em três passos: escrever `feval.in`, executar o script `airfoil.py` e ler `feval.out`. Não utilizamos restrições por enquanto, exceto as de caixa. Nenhum passo adicional é necessário.

6 Otimização de aerofólios

Manteremos fixos os seguintes parâmetros: $\alpha = 10°$, $U_\infty = 291.55\text{m/s}$, $N_1 = 0.5$, $N_2 = 1$, $d = 2$, $\zeta = 0$, $C_{\text{middle,trail,lead}} = (0.05, 0.15, 0.003)$, e todos os outros parâmetros conforme o padrão. Resolvemos o seguinte problema de otimização:

$$\text{Minimizar } f(A_U, A_L) \doteq \frac{C_D}{C_L}$$
$$\text{sujeito a } \ A_U, A_L \in [0, 2]^3,$$

A solução encontrada pelo ALGENCAN a partir de um ponto inicial aleatório é dada por $A_u^* = (0.1889, 0.3828, 0.0455)$ e $A_l^* = (0.0001, 0.0386, 0.001)$ e apresenta $C_D = 0.0076$ e $C_L = 0.3101$. Visualisando-a no PARAVIEW, juntamente com a velocidade e pressão computadas pelo OPENFOAM, temos o seguinte:

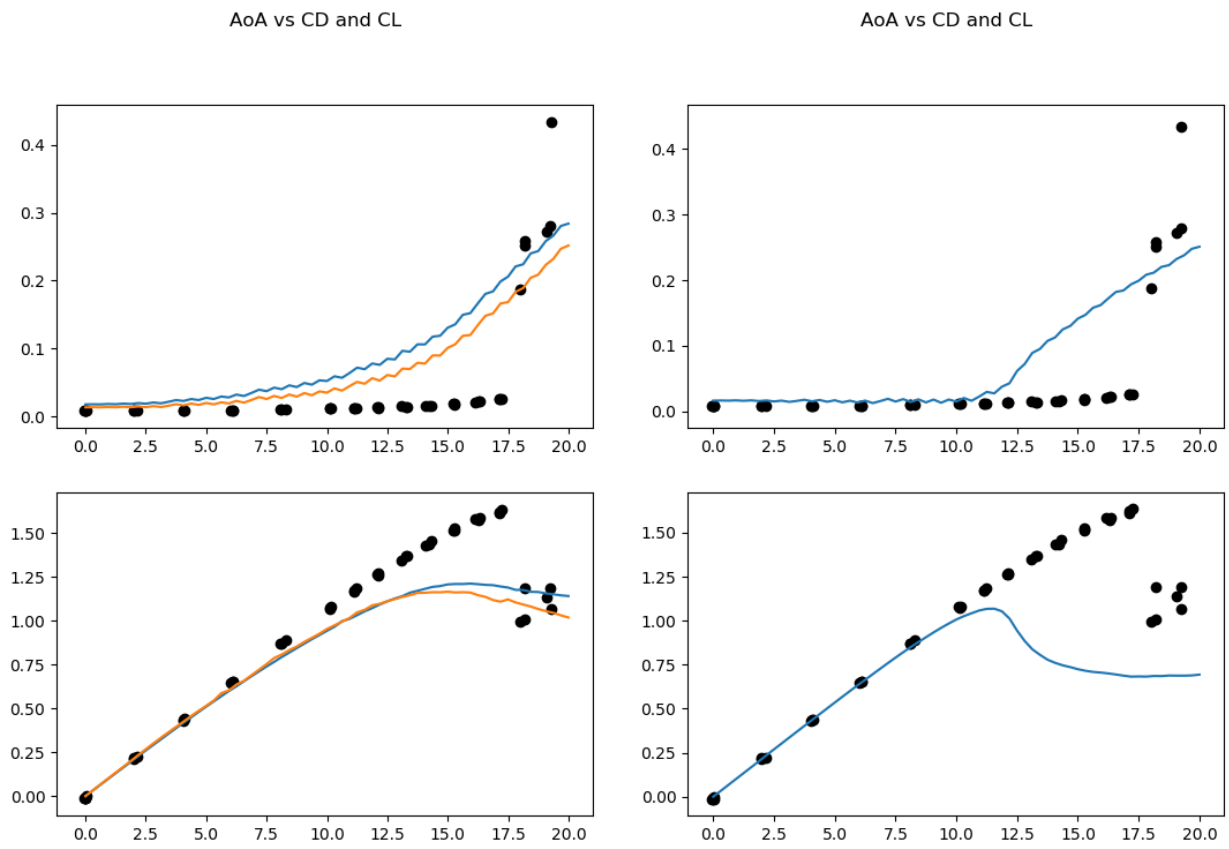


O ALGENCAN convergiu pelo resíduo e é possível verificar que, de fato, o aerofólio encontrado é um ótimo local de f.

6.1 Validação dos resultados

Para que possamos confiar que, de fato, o aerofólio obtido é um aerofólio ótimo real, é necessário ter certeza de que f está sendo computada propriamente. Isso é feito por meio de uma validação computacional em comparação com dados experimentais reais. Dado que não é possível (para nós) construirmos o aerofólio obtido pelo ALGENCAN e testá-lo numa simulação real, fizemos esta validação por meio de um aerofólio conhecido como NACA0012. Tais dados experimentais, juntamente a uma malha de validação, podem ser encontrados no site do OPENFOAM.

Utilizamos um método de discretização do tipo *upwind* (e também o *linear upwind*), o que pode ter algum efeito sobre a qualidade da solução. Além disso, os experimentos foram feitos com duas malhas menos refinadas que as fornecidas pelo OPENFOAM (em azul e laranja na figura abaixo). Obtivemos o seguinte:



Os gráficos da parte de cima representam $\alpha \times C_D$ e na parte de baixo $\alpha \times C_L$. Na figura da esquerda, temos o teste com *upwind* e na figura da esquerda, com *linear upwind*. Os pontos em preto correspondem aos valores obtidos experimentalmente. Vemos que a simulação numérica é razoavelmente fiel à realidade para $\alpha \in [0, 10]$, mas discrepante com $\alpha > 10$. A partir deste ponto, é natural inferir que o resultado do processo de otimização é confiável, dado que simulamos o problema com $\alpha = 10$. Porém, um olhar mais crítico nos faz perceber que não necessariamente o intervalo de confiabilidade de α para o aerofólio obtido pelo Algencan é o mesmo do NACA0012. Portanto, a corretude da solução obtida ainda não está clara.

7 Considerações finais

Nós integramos o software de dinâmica dos fluidos OpenFoam com o software de otimização não linear Algencan por meio de uma interface de automatização de uso do OpenFoam. A partir disso, o problema de otimização do aerofólio proposto pôde ser resolvido numericamente. Porém, ainda há questões em aberto sobre a confiabilidade da solução. Para que a solução encontrada seja válida, é necessário validar o modelo e sua escolha de parâmetros comparando os dados simulados numericamente com dados experimentais. Demos passos nesta direção, mas não obtivemos evidências fortes o suficiente para que possamos considerar o aerofólio obtido, ótimo.

A principal tarefa para qualquer pessoa que desejar dar continuidade a este trabalho é validar

o modelo de maneira satisfatória. Não recomendamos prosseguir em outras áreas sem que este ponto esteja perfeitamente fechado. Estando esta questão resolvida, pode-se pensar em diferentes modelos de otimização para resolver o problema, ou mesmo na aceleração do método. Para resolver o problema de validação, nós recomendamos investigar mais a fundo os métodos de resolução de subproblemas da rotina `simpleFoam` a fim de encontrar aquele que melhor se aplica ao nosso problema.

Referências

[1] D. Collela, F. Gritten, F. López, R. Moreira Introdução ao Design de Asas.

[2] H. Versteeg, W. Malalasekera. An Introduction to Compitational FLuid Dynamics. Prentice Hall. 2007, 2nd edition.

- Livros e artigos oficiais na web:
  - Rist, Ian ; McGinnigle, James. Tensors in Materials Science. DoITPoMS, University of Cambrige. 2008. Disponível em: <https://www.doitpoms.ac.uk/tlplib/tensors/what_is_tensor.php>. Acesso em: 25 de Nov de 2021.
  - P.R. Spalart; S.R. Allmaras. A one-equation turbulence model for aerodynamic flows. 1994. Disponível em <https://www.researchgate.net/publication/236888804_A_One-Equation_Turbulence_Model_for_Aerodynamic_Flows>. Acesso em: 05 de Dec de 2021.
- Web-sites:
  Observação: Alguns dos web-sites embaixo, correspondem a introdução de figuras, tanto deste relatório final, como das presentações correspondentes a este projeto.
  - https://turbmodels.larc.nasa.gov/spalart.html.
  - https://www.guiadaengenharia.com/numero-reynolds-entenda/
  - https://pt.wikipedia.org/wiki/Turbul%C3%AAncia
  - https://sites.google.com/site/scientiaestpotentiaplus/gradiente-de-pressao-advers
  - http://www.estruturas.ufpr.br/material/mecanicaContinuo/campos.html
- Figuras:
  - https://pt.khanacademy.org/science/physics/fluids/fluid-dynamics/a/what-is-bernou
  - https://wikiciencias.casadasciencias.org/wiki/index.php/Viscosidade
  - https://www.123calculei.com/Cubo/
  - https://pt.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/divergence-and-curl-articles/a/divergence
  - https://www.scielo.br/j/rbef/a/tsvTgNcxwdxScrcCQtznGkK/?lang=pt&format=pdf

- Vídeos:
  Observação: Os seguintes não foram usados como referências, porém, as explicações dadas nos vídeos abaixo, podem ajudar ao leitor a entender os conceitos implicados neste texto pois foram de gram ajuda na compreensão dos termos, e as explicações dos conceitos presentes neste texto.

  - CFD (*Computational Fluid Dynamics*)
    * CFD Simulation of Airfoil | Tutorial | Concepts
      https://www.youtube.com/watch?v=20Kq04pmczs
    * COMPUTACIONAL FLUID DYNAMICS | CFD BASICS
      https://www.youtube.com/watch?v=hXG1uCiIe-U&list=PLnrE77fLDvaxeqsHYPYcnnatmiVS
      index=7&t=266s
  - Da equação de Bernoulli:
    * Equação de Bernoulli - Teoria, exemplos e resolução de exercício
      https://www.youtube.com/watch?v=4-QjbYd4aqI&list=PLnrE77fLDvaxeqsHYPYcnnatmiVS
      index=1&t=28s
  - Da equação de Navier-Stokes:
    * Cómo entender por primera vez la ecuación de Navier-Stokes?
      https://www.youtube.com/watch?v=9WfsWRlMcQQ&list=PLnrE77fLDvaxeqsHYPYcnnatmiVS
      index=4
    * Introdução às Equações de Navier-Stokes: perspectiva histórica, conceitos fundamentais e aplicações
      https://www.youtube.com/watch?v=svh7UGIqrYQ&t=4466s
    * Deduzindo Navier-Stokes de verdade
      https://www.youtube.com/watch?v=OP6e1ERJrqI
    * Notação Indicial - Parte 1
      https://www.youtube.com/watch?v=54IltZnHifw&t=2247s
    * Equação de Navier-Stokes (Parte 1) - Derivadas materiais
      https://www.youtube.com/watch?v=FLoZODPpayM
    * Equação de Navier-Stokes (Parte 2) - Equação diferencial da quantidade de movimento
      https://www.youtube.com/watch?v=e06ZRzdO4iM
    * Equação de Navier-Stokes (Parte 3)- Tensões Normais e Cisalhantes
      https://www.youtube.com/watch?v=na2kGOSYNv8
    * Equação da Continuidade e equação de Cauchy
      https://www.youtube.com/watch?v=W36ONpadK_g&t=1724s
  - Da turbulencia:
    * CFD El modelo de turbulencia de Spalart-Allmaras
      https://www.youtube.com/watch?v=XivcOEIGFQw&t=254s

A Instruções de instalação para leigos

A primeira e mais importante regra é: não use Windows, a menos que você já tenha muita experiência com programação em Windows. Se você for um programador medíocre, assim como nós, use qualquer distribuição do Linux.

A.1 Instalação do OpenFOAM

Para instalar o OpenFOAM, basta seguir as instruções da Wiki oficial: https://develop.openfoam.com/Development/openfoam/-/wikis/precompiled/. Basicamente, você só tem que abrir o terminal e digitar duas coisas:

```
wget -q -O - https://dl.openfoam.com/add-debian-repo.sh | sudo bash
```

```
sudo apt-get install openfoam2106-default
```

Confirme tudo o que tiver que confirmar. O OpenFOAM provavelmente foi instalado na pasta `/usr/lib/openfoam/openfoam2106`. O que você deve fazer agora para usar o OpenFOAM é "indexá-lo", para isso, abra o seu bash com `gedit ~/.bashrc`, vá até a última linha do arquivo que abriu, e inclua o seguinte:

```
source /usr/lib/openfoam/openfoam2106/etc/bashrc
```

Agora, para testar o OpenFOAM, copie a pasta `/openfoam2106/tutorials` para algum outro lugar do seu computador, e vá até `/tutorials/incompressible/simpleFoam/airFoil2D`. Digite `./Allrun` e depois `simpleFoam`. Se nada der errado, ótimo.

A.2 Instalação do ParaView

Esse ParaView é uma ferramenta de visualização, por enquanto. Todas as operações de verdade serão feitas no OpenFOAM. Para instalar o ParaView, basta pegar do repositório:

```
sudo apt-get install paraview
```

Digite `paraview` para testar. Na pasta `airFoil2D`, tente rodar `paraFoam` depois de `simpleFoam`. Se funcionou, pronto. Instalado.

A.3 Instalação do Algencan

É possível fazer download do Algencan diretamente da página do Projeto TANGO mantida por Ernesto G. Birgin, basta deixar seus dados. Após o download, extraia a pasta do Algencan no lugar onde o seu projeto principal se encontra, entre nela pelo terminal e digite:

```
sudo make
```

Quando o `make` terminar, você deve copiar o arquivo gerado `libalgencan.a` para a pasta `lib` do seu sistema[2] com:

[2]**Nota dos autores:** Bom, nós tivemos que fazer isso, mas as instruções do Ernesto dizem que não precisa.

```
sudo cp libalgencan.a $ALGENCAN/lib/
```

Depois disso, você poderá chamar o ALGENCAN de qualquer lugar. Para gerar um arquivo de entrada para ele, pegue qualquer exemplo da pasta `sources/examples`, sendo o mais simples deles o `toyprob.f90`, e mude a função objetivo e as restrições para o que você quiser. Para rodar o programa, basta compilar e executar da maneira usual; por exemplo, para o `toyprob.f90`, digite:

```
gfortran -O3 toyprob.f90 -L$algencan-3.1.1/lib -lalgencan -o toyprob
```

E depois execute-o com

```
./toyprob
```

Agora, basta modificar o `toyprob.f90` com as funções e restrições do seu interesse para resolver o seu problema. Para ter acesso a um manual bem didático do ALGENCAN, nós recomendamos um texto de Rian Penachi intitulado "*Uma introdução à otimização não linear e a solução de problemas simétricos via* ALGENCAN".