

Universidade Federal do Rio de Janeiro



Estudos sobre simulação computacional

Método dos Elementos Finitos

Solução de problemas de transferência de calor e mecânica dos
fluidos

Aluno
Professor

Matheus Lira Sartor
Gustavo Rabello

Rio de Janeiro, 18 de outubro de 2021

Conteúdo

1	Introdução	1
1.1	Método dos Elementos Finitos	2
1.1.1	O método	3
1.1.2	Implementação das condições de contorno e solução do problema	5
2	Exercício 1 - Transferência de calor em regime transiente	8
2.1	Resultados da Simulação	11
3	Exercício 2 - Função Corrente	15
3.1	A função corrente	15
3.2	Resultados da Simulação	17
4	Exercício 3 - Função Corrente-Vorticidade	20
4.1	Resultados da simulação	22
4.1.1	Pequeno cilindro entre paredes planas	22
4.1.2	Escoamento numa cavidade	23
4.1.3	Escoamento no canal com abertura	25
5	Conclusão	26
6	Referências	27
A	Apêndice - Códigos	28
A.1	Simulação do problema de transferencia de calor transiente . .	28
A.2	Simulação do problema da função corrente	33
A.3	Simulação do problema da função corrente-vorticidade	34

1 Introdução

Realizar simulações é uma tarefa que está cada vez mais presente no dia-a-dia do profissional de engenharia qualificado. Isto ocorre simplesmente pois o custo com simulações costuma ser bem menor que o custo com repetidos testes. Além do mais, as simulações são importantes aliadas ao desenvolvimento científico, visto que a partir delas pode-se ter uma ideia de como ocorrem determinados fenômenos sem necessariamente dispor de uma bancada de ensaio. Há também necessidade de resolver problemas de mecânica dos fluidos e transporte de calor para otimizar os processos industriais ou desenvolver novas tecnologias. Pode-se confabular sobre um projeto de placa mãe, para o funcionamento correto do computador. É necessário que os componentes da placa mãe não superaqueçam e acabem tendo seu funcionamento comprometido.

Para isso, existem várias formas de resfriar o ambiente ou ajudar a dissipar esse calor gerado pelos componentes de desempenho da máquina, como memórias RAM, processadores, SSD e GPU; desde o uso de *fan coolers* como alterando a geometria dos dissipadores de calor dos componentes, se utilizando de aletas, por exemplo, mas isto fica a critério do projetista. Portanto, para o projeto de uma placa mãe pode se tornar de extrema valia a realização de simulações para conferir a temperatura que os componentes podem atingir para então, verificar a necessidade de mudanças no projeto e a eficácia delas. Desta forma, o projetista consegue ter uma estimativa muito boa da eficiência dos processos sendo utilizados para resfriamento da placa antes mesmo de comprá-los e assim ter um projeto de placa mãe mais confiável e possivelmente menos custoso.



(a) Placa Mãe Asus
Prime A320M



(b) Cooler AMD Intel Ice Edge Mini FS V2.0

Figura 1: Representação gráfica de uma placa mãe e um cooler. Fonte: Amazon

1.1 Método dos Elementos Finitos

O método dos elementos finitos (MEF) surgiu pela quinta década do século XX, quando foram lançados os primeiros computadores. Os fundamentos matemáticos do MEF já eram conhecidos havia tempo, mas as ferramentas de cálculo então disponíveis inviabilizavam a sua implementação e utilização.

Inicialmente o MEF foi aplicado na análise de problemas da mecânica dos sólidos, mas logo a sua aplicação estendeu-se ‘a análise de outros fenômenos físicos. Esta abrangência mais o sucesso do método propiciaram o estudo mais profundo e extenso dele. Da análise matemática do método resultaram estimadores de erro e critérios de estabilidade, que garantem aos resultados mais confiabilidade. Da análise estática passou-se à dinâmica; dos problemas inicialmente lineares passou-se aos não-lineares; da análise de um único fenômeno passou-se à de vários fenômenos simultâneos e interagentes; de interfaces computador-usuário pouco práticas passou-se às interfaces gráficas, mais amigáveis e intuitivas... No presente o MEF continua evoluindo nos seus diversos aspectos, conforme demonstra a quantidade de artigos científicos atualmente publicados em torno dele.

A beleza da engenharia está não só no conhecimento multidisciplinar de matemática e outras ciências da natureza, mas também em unir estes conceitos para a realização de um projeto ou a resolução de um problema que represente a realidade de forma mais aproximada possível. Para isso, o profissional de engenharia deve estar apto a construir modelos que digam quais são as informações necessárias e quais são as variáveis a se calcular. Primeiramente então se analisa a realidade, a partir dela cria-se um modelo físico, que pode levar a um modelo matemático que, por sua vez, pode ser solucionado numericamente.



Figura 2: Sequência de modelos

Uma abordagem muito utilizada para análise de fluido-dinâmica e problemas de transferência de calor é a definição de uma região de controle, a partir da qual se definem condições de contorno e condições iniciais no caso de regime transiente. A partir destas informações é possível determinar como os parâmetros de velocidade e temperatura irão se comportar na região utilizando-se de equações diferenciais parciais.

Estas equações, mais conhecidas como EDPs, basicamente modelam como determinada propriedade irá variar no espaço e no tempo de acordo com os parâmetros de entrada e podem ser definidas basicamente como balanços de massa e energia. Em regime permanente, por exemplo, a lógica que pode ser implementada é que tudo aquilo que entra na região de controle deve sair em igual quantidade para que o sistema permaneça em equilíbrio.

Agora imagine um problema térmico, como o que foi mostrado da placa mãe anteriormente, uma vez que sabe-se que no regime permanente é atingido o equilíbrio térmico e as trocas de calor com o meio externo não afetarão mais a temperatura, então desde que se saiba a temperatura em lugares específicos e a quantidade de calor conduzida no meio, pode-se então determinar o estado final das temperaturas no corpo. Estes lugares específicos podem ser uma ou mais paredes da região de controle. E essas condições que definirão o problema podem ser chamadas de condições de contorno. No escopo deste documento serão apresentadas explicações sobre as condições de contorno de Neumann e de Dirichlet.

1.1.1 O método

Como forma de introdução, será traduzida aqui uma equação de transferência de calor e em seguida serão montadas as matrizes a serem utilizadas na solução do problema. A equação a ser utilizada será a seguinte:

$$\frac{\partial(\rho c_v T)}{\partial t} = \nabla \cdot k \nabla T + Q \quad (1)$$

A equação (1) acima, que é considerada a forma forte da equação de calor transiente, significa que a temperatura irá variar no tempo numa taxa proporcional ao produto do calor específico do material e da densidade, ao passo que a análise quantitativa dessa variação é de responsabilidade do parâmetro k , que é a condutividade térmica. Caso deseje analisar o resultado em regime permanente, basta zerar o lado esquerdo da equação.

O primeiro passo para a aplicação do método dos elementos finitos é então a multiplicação pela função peso $w(x, y, z)$ e então a integração no domínio $\Omega = dx dy dz$ (O exemplo aqui tratado é tridimensional, entretanto o mesmo vale para modelos bidimensionais).

$$\int_{\Omega} w \left(\frac{\partial(\rho c_v T)}{\partial t} - \nabla \cdot k \nabla T - Q \right) d\Omega = 0 \quad (2)$$

Desmembrando os termos, obtem-se:

$$\int_{\Omega} w \frac{\partial(\rho c_v T)}{\partial t} d\Omega - \int_{\Omega} w \nabla \cdot k \nabla T d\Omega - \int_{\Omega} w Q d\Omega = 0 \quad (3)$$

Agora utilizando a identidade de Green para os campos com escalares, no caso do ∇ . pode-se desacoplar o segundo termo da equação:

$$\int_{\Omega} w \frac{\partial(\rho c_v T)}{\partial t} d\Omega - \int_{\Gamma} w k \nabla T d\Gamma - \int_{\Omega} \nabla w \cdot k \nabla T d\Omega - \int_{\Omega} w Q d\Omega = 0 \quad (4)$$

Onde o domínio Γ representa o contorno da região de controle. Nas condições de contorno de Dirichlet, que serão vistas posteriormente, este termo pode ser igualado a zero.

O próximo passo é definir as funções peso $w(x, y, z)$ e temperatura $T(x, y, z)$ como uma combinação linear, como tentativa de linearização do problema. Pode-se discretizar também os termos escalares, como Q , da seguinte forma:

$$T(x, y, z) = \sum_{i=0}^{\infty} N_i(x, y, z) a_i; \quad w(x, y, z) = \sum_{j=0}^{\infty} N_j(x, y, z) b_j$$

$$Q(x, y, z) = \sum_{i=0}^{\infty} N_i(x, y, z) Q_i$$

A maneira mais simples de resolver essas discretizações é utilizando o método de Galerkin, que consiste em utilizar funções de forma N_i e N_j que sejam iguais. Como não se tem infinitos pontos na malha também supõe-se que a aproximação sirva para o numero de pontos da malha. ou seja, quanto mais cheia a malha, provavelmente mais próximo do resultado real será o resultado da simulação, entretanto, maior será o custo computacional também.

Ao substituir as funções de T , w e Q pela forma discretizada, obtém-se o seguinte resultado:

$$\sum_{i=0}^n \sum_{j=0}^n \int_{\Omega} N_i N_j \frac{da_i}{dt} d\Omega + \sum_{i=0}^n \sum_{j=0}^n \left(- \int_{\Gamma} w (k \nabla N_i) d\Gamma + \int_{\Omega} k \nabla N_i \cdot \nabla N_j d\Omega \right) a_i$$

$$= \int_{\Omega} N_i N_j Q d\Omega$$

Se o termo no domínio do contorno for anulado (no caso de Dirichilet ou Neumann). Tem-se uma equação com termos que se repetem.

$$\sum_{i=0}^n \sum_{j=0}^n \int_{\Omega} N_i N_j \frac{da_i}{dt} d\Omega + \sum_{i=0}^n \sum_{j=0}^n \left(\int_{\Omega} k \nabla N_i \cdot \nabla N_j d\Omega \right) a_i$$

$$= \int_{\Omega} N_i N_j Q d\Omega$$

Destes termos em evidência, pode-se extrair o seguinte:

1. $\nabla N_i \cdot \nabla N_j = \mathbf{K}$ que será a matriz de rigidez do sistema.
2. $N_i \cdot N_j = \mathbf{M}$ que será a matriz de massa do sistema.

As matrizes \mathbf{M} e \mathbf{K} dependem das funções de forma aplicadas (se é linear, quadrática ou quaisquer outras funções) e da geometria do elemento, visto que as funções N_i e N_j precisarão atender o seguinte critério:

- N_i e N_j devem valer 1 em um ponto e 0 nos adjacentes

Agora, com a equação da temperatura discretizada (forma fraca), pode-se resolver o problema térmico através da solução do sistema linear mostrado abaixo:

$$\rho c_v \mathbf{M} \frac{dT}{dt} + k \mathbf{K} T = \mathbf{M} Q + c.c. \quad (5)$$

Onde c.c. são as condições de contorno, que farão parte do problema e serão cruciais para a solução do mesmo.

1.1.2 Implementação das condições de contorno e solução do problema

Agora que sabe-se a equação diferencial parcial em sua forma fraca, é possível transformar esta equação em um sistema linear simplificado, no caso de regime permanente, o termo $\frac{dT}{dt}$ irá ser nulo e portanto, para calcular o valor das temperaturas para todos os pontos, basta fazer o seguinte:

$$k \mathbf{K} T = \mathbf{M} Q$$

isso se assemelha a um sistema $\mathbf{A}x = \mathbf{b}$, onde $\mathbf{A} = k\mathbf{K}$, $\mathbf{b} = \mathbf{M}Q$ e x é o vetor de temperaturas desejado. Sendo assim:

$$\begin{aligned} T &= (k\mathbf{K})^{-1} \mathbf{M} Q \\ T &= \mathbf{A}^{-1} b \end{aligned}$$

Hmmm, certo. Mas e as condições de contorno? como aplicá-las? Muito simples! A própria solução deste sistema linear já subentende que a variável desejada não irá variar no sentido da direção normal ao calculado, ou seja, $\nabla T \cdot \mathbf{n} = 0$. Logo, a equação nesta forma estará aplicando a condição de contorno de Neumann para todo o domínio do contorno; entretanto, pode ser de desejo do projetista aplicar condições de contorno de Dirichlet. Neste caso, o procedimento a ser realizado é o seguinte:

1. Antes de inverter a matriz \mathbf{A} para resolver o sistema, será necessário identificar quem são os nós de contorno onde será aplicada a condição de Dirichlet.
2. Após identificar os nós de contorno, deve-se zerar a linha correspondente a eles na matriz \mathbf{A} e então deve ser colocado o valor 1 na posição que seria a diagonal da matriz na linha correspondente.
3. Feito isso, basta intão aplicar o valor conhecido da condição de Dirichilet ao vetor b , na posição do nó de contorno.
4. Agora sim, inverter a matriz \mathbf{A} e solucionar o sistema linear com as condições de contorno aplicadas.

Além das matrizes de massa e de rigidez, pode surgir também a matriz de gradiente \mathbf{G} , que nada mais é que o produto $N.\nabla N$ ou $\nabla N.N$, ela será de suma importância ao investigar parâmetros de convecção, pois ela dá uma ideia de como a malha varia em função dos eixos ortogonais.

Na explicação acima, foi solucionado o problema considerando o regime permanente, mas e no caso onde haja interesse em saber como essa transferência de calor afeta a temperatura ao longo do tempo? Neste caso, não se deve zerar o termo temporal, entretanto, pode-se discretizá-lo como é feito no método de diferenças finitas:

$$\rho c_v \mathbf{M} \frac{T^{n+1} - T^n}{\Delta t} + k \mathbf{K} T = \mathbf{M} Q$$

Onde o vetor T^{n+1} é o T futuro, desconhecido, enquanto o T^n é o valor de T atual, conhecido. Desta forma, existem algumas formas de resolver este sistema, elas dependem de como pretende-se tratar o vetor T que acompanha a matriz de rigidez \mathbf{K} . Caso queira tratá-lo como o T atual, conhecido, então este é conhecido como método explícito. Caso queira tratá-lo como como o vetor T futuro, desconhecido, isto caracteriza a solução como método implícito. Existem uma possibilidade também de obter combinações destes métodos, afim de extrair as vantagens dos dois métodos. Geralmente, o método implícito necessita de uma maior quantidade de recursos computacionais e um maior numero de iterações, ao passo que o método explícito sofre com limitações de extrapolações quanto ao passo de tempo (só funciona para Δt pequeno o suficiente). Pode-se adicionar um parâmetro θ para definir o método a ser utilizado na solução do sistema, a equação com θ fica da seguinte forma:

$$\rho c_v \mathbf{M} \frac{T^{n+1}}{\Delta t} + \theta k \mathbf{K} T^{n+1} = \rho c_v \mathbf{M} \frac{T^n}{\Delta t} - (1 - \theta) k \mathbf{K} T^n + \mathbf{M} Q$$

$$\left(\rho c_v \frac{\mathbf{M}}{\Delta t} + \theta k \mathbf{K} \right) T^{n+1} = \left(\rho c_v \frac{\mathbf{M}}{\Delta t} - (1 - \theta) k \mathbf{K} \right) T^n + \mathbf{M} Q$$

Para simplificar a equação, podemos ainda afirmar que há um parâmetro $\alpha = \frac{k}{\rho c_v}$ e então dividir ambos os lados da equação por ρc_v , obtém-se então o seguinte resultado:

$$\left(\frac{\mathbf{M}}{\Delta t} + \theta \alpha \mathbf{K} \right) T^{n+1} = \left(\frac{\mathbf{M}}{\Delta t} - (1 - \theta) \alpha \mathbf{K} \right) T^n + \mathbf{M} Q_k \quad (6)$$

Onde $Q_k = \frac{Q}{\rho c_v}$.

No método dos elementos finitos, estes parâmetros de densidade, condutividade térmica e calor específico do material são facilmente atribuídos às regiões com os materiais desejados, bastando apenas acoplá-los às matrizes de massa e rigidez nesse caso. Ao percorrer a matriz de conectividade, basta definir qual é o material para cada elemento conforme a região definida por ele. A matriz de conectividade e as regiões da geometria para a simulação podem ser configuradas em seu gerador de malhas favorito. Para o atual objeto de estudo, foi utilizado o editor em código livre GMsh.

Quanto aos valores de θ , é possível obter soluções utilizando valores de 0 a 1, onde estes valores representam o seguinte:

- $\theta = 0$: Método explícito
- $\theta = 1/2$: Método de Crank-Nicholson
- $\theta = 1$: Método implícito

O método de Crank-Nicholson ao iterar sobre as matrizes irá utilizar parcialmente a matriz de rigidez para a parte implícita (desconhecida) da equação e outra parte para a parte explícita. Essa combinação pode gerar resultados intermediários como, por exemplo, quando certo Δt é desejado, entretanto este extrapola os limites do método explícito e não se tem disponibilidade computacional para resolver o problema através do método implícito. Cabe ao profissional de engenharia enxergar essas nuances e decidir qual pe o melhor valor a se utilizar.

2 Exercício 1 - Transferência de calor em regime transitente

O objeto de estudo para este exercício foi apresentado no início do texto, trata-se da placa mãe ASUS Prime A320M K/BR Amd AM4 DDR4. O modelo dispõe de 1 *slot* para processador, 2 barramentos para memória RAM, um barramento para o SSD e 1 *slot* para GPU, os demais componentes eletrônicos como resistores e capacitores foram ignorados, com exceção da fonte de alimentação. Para a simulação foi estimado um consumo de 65W para a placa, e destes, foi estimado que 60% seriam transformados em calor para os componentes. Abaixo pode ser observada a visão superior da placa, a qual serviu de inspiração para a criação da malha.

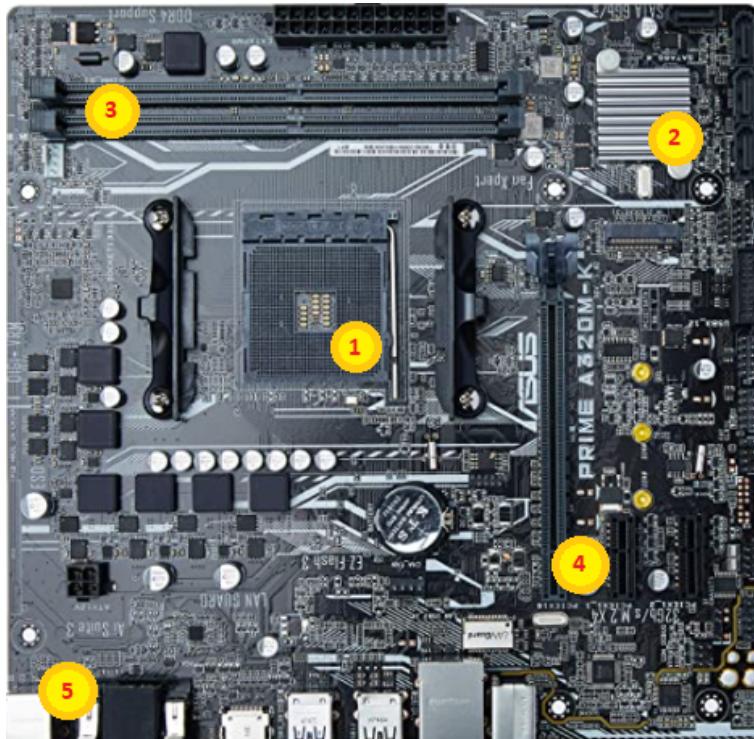


Figura 3: Vista superior da placa mãe Asus Prime A320M-K

1. Espaço para o processador
2. Dissipador de calor para a conexão com a placa de vídeo
3. Barramentos para encaixe dos pentes de memória RAM
4. Barramento para encaixe do SSD

5. Entrada de alimentação

A partir da imagem, foram recolhidas as medidas e através do programa GMsh foi construída a malha que irá ser utilizada para as simulações.

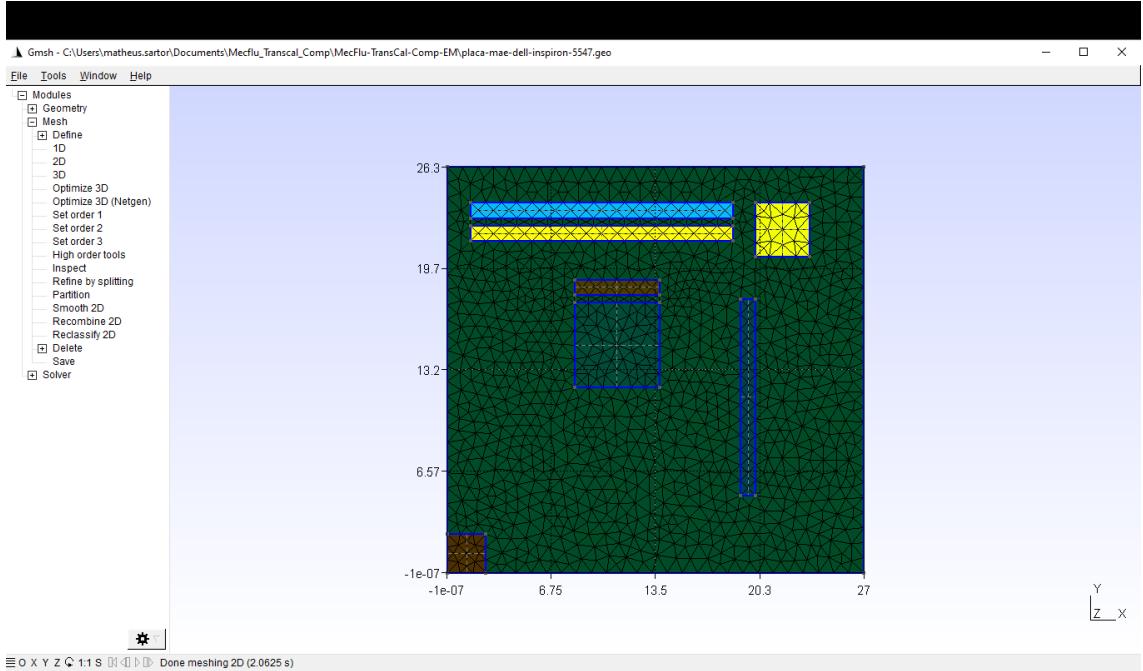


Figura 4: Malha para a placa construída através do programa GMsh

Como é possível observar, o GMsh permite que divida-se a malha em diferentes regiões, sendo cada uma delas uma superfície, para este caso 2D. Na figura, as regiões ficam denotadas por cores diferentes. Essas configurações de visualização são configuráveis no próprio aplicativo.

A equação utilizada no método dos elementos finitos para a malha foi a mesma equação (6), mostrada anteriormente, e os elementos utilizados foram os triângulos lineares, ou seja, uma malha triangular utilizando a equação da reta como N_i e N_j para interpolação entre os pontos. Como foi utilizado o método de Galerkin, $N_i = N_j$, e portanto, as matrizes de massa e rigidez podem ser obtidas de duas maneiras:

1. Encontrando as funções N_i, N_j que satisfazem a condição de valor 1 no ponto e 0 nos demais e então encontrando o resultado de $N_i \cdot N_j$ para a matriz \mathbf{M} e $\nabla N_i \cdot \nabla N_j$ para a matriz \mathbf{K}

- A outra forma é verificar em algum *cheatsheet* se há menção ao elemento que será utilizado. Visto que para o mesmo tipo de elemento com o mesmo tipo de interpolação, as matrizes serão as mesmas.

Para este caso de estudo, as matrizes utilizadas foram obtidas através da folha de dicas (ver [4], p.35), na descrição do triângulo linear.

A malha foi então dividida em 8 regiões, sendo duas delas comportando o processador, duas para os barramentos de RAM, uma para a placa de vídeo, uma para o barramento do SSD e uma para a fonte de alimentação.

Como dito anteriormente, foi considerado um consumo de 65W (consumo médio de um notebook, ver [5]) E uma estimativa de 60% desta energia sendo dissipada como calor para a placa. Sendo assim, destes 60%, um quarto foi considerado gerado pelo processador, outro quarto gerado pela placa de vídeo e o restante distribuído igualmente entre os componentes de memória e alimentação.

A partir daí, obteve-se através do apêndice do livro do Ozisik [6], os valores para os parâmetros de materiais a serem utilizados. Os materiais utilizados foram:

- Lã de vidro: Como isolante para a base da placa.
- Alusil (Liga de Al-Si): para as entradas do processador e da placa de vídeo
- Cobre: Para as conexões dos barramentos de RAM e SSD
- Alumínio: Para a entrada da fonte de alimentação

As propriedades recolhidas podem ser visualizadas na tabela a seguir (optou-se por converter as propriedades para cm por conta do valor de α , que se utilizado em $\frac{m^2}{s}$ pode atingir numeros na ordem de 10^{-7} , podendo, no processo das contas, gerar zeros numéricos para o computador comum):

	$\rho \left[\frac{g}{cm^3} \right]$	$c_v \left[\frac{J}{gC} \right]$	$\kappa \left[\frac{W}{cmC} \right]$
Alusil	2,659	0,867	1,61
Cobre	8,960	0,383	3,86
Aluminio	2,707	0,921	2,04
Lã de Vidro	1,790	0,712	0,22

Tabela 1: Propriedades dos materiais utilizados no ensaio

2.1 Resultados da Simulação

Utilizando a biblioteca matplotlib, é possível visualizar os resultados, frame a frame, da simulação, como amostra animada, foi realizada uma simulação com passo de tempo grande para a fácil observação do desenvolvimento térmico da placa durante um minuto de funcionamento, sem nenhum dissipador ou cooler para melhorar a distribuição do calor. Neste caso, foi utilizado o método implícito, para não sofrer problemas de extração do tempo.

É recomendável que o leitor utilize o Adobe Acrobat Reader DC ao visualizar este documento para conseguir visualizar o resultado da simulação de maneira dinâmica.

De qualquer forma, uma outra simulação foi realizada, utilizando os mesmos parâmetros e um passo de tempo menor, para uma melhor observação

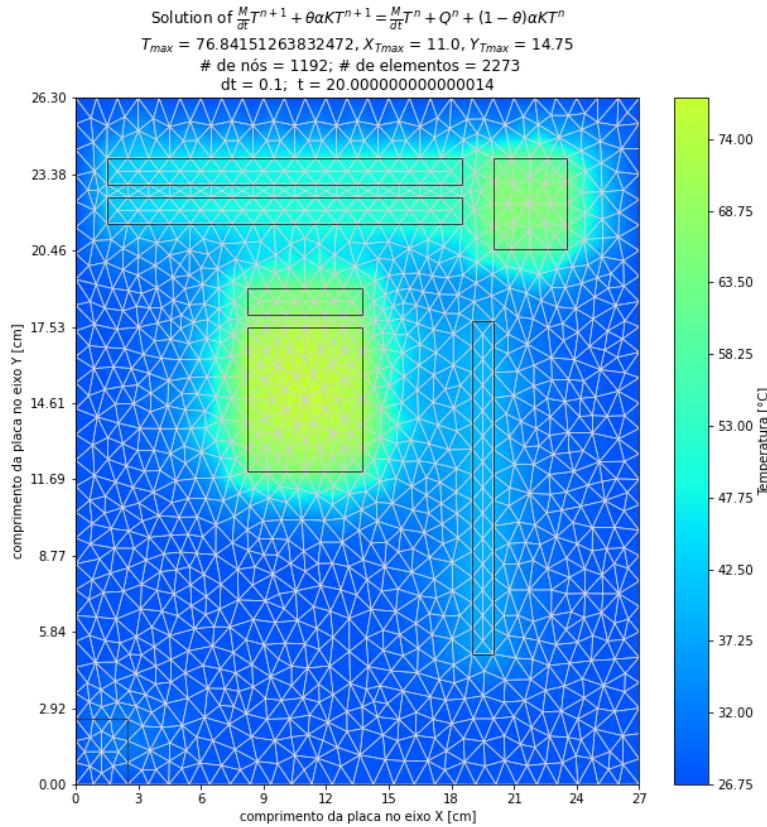


Figura 5: Resultado em $t = 20$, $dt = 0.1s$

da influência da geração de calor nas mudanças de temperatura.

depois de 300 iterações no tempo, com um passo de 0.1 segundo, obteve-se então o seguinte resultado:

Para checar a solução em regime permanente, basta zerar os termos contendo dt , logo, em regime permanente, o gradiente de temperaturas fica dessa forma:

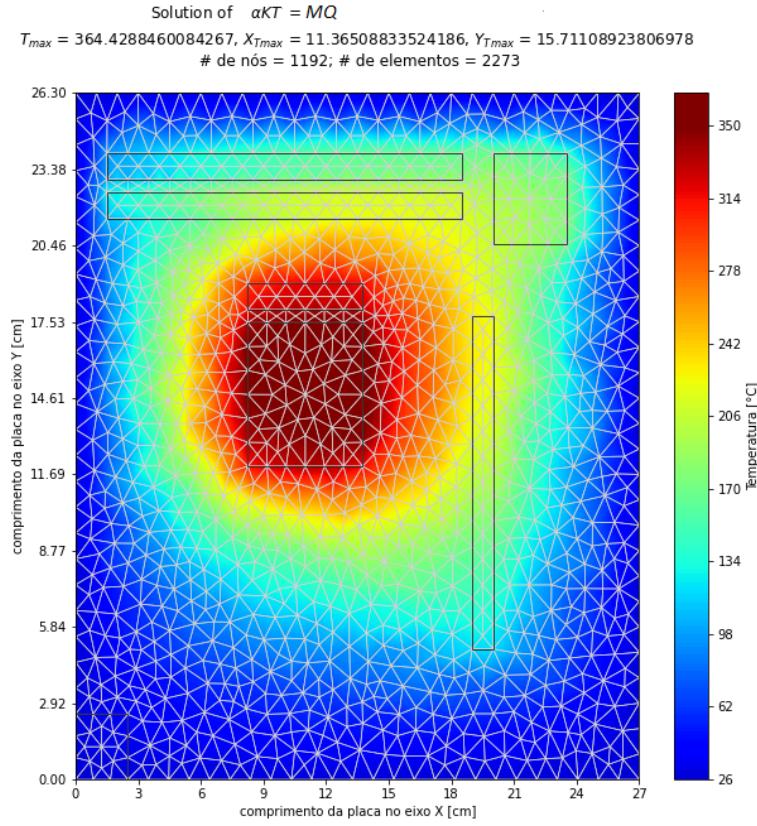


Figura 6: Resultado em regime permanente

Uau! Trabalhar a 365°C parece algo bem improvável para uma máquina com tantos componentes sensíveis como o computador pessoal. Segundo a estudante de Ciência e Tecnologia na UFRGS Priscilla Kinast[7], a temperatura normal de trabalho de um processador pode chegar até 90°C , sendo que a maioria das CPUs começará a acelerar (diminuir sua velocidade de clock para reduzir o calor) quando atingirem de 95 a 105°C . Se as temperaturas aumentarem ainda mais, a CPU se desligará, para evitar danos permanentes.

Isso significa que provavelmente os valores para geração de calor não chegam a tanto normalmente, ou então que o sistema não está corretamente refrigerado (o que é uma clara verdade). Não se deve também descartar a hipótese de que este modelo é deveras simplificado por estar deixando de considerar a dissipação de calor por convecção pelo ar e estar apenas considerando as condições de Dirichlet na parede, com uma temperatura inicial de 27°C e temperaturas constantes de 27°C no contorno da placa. Caso o

problema seja ampliado para simulações em 3 dimensões, haverá também a possibilidade do uso de aletas para a dissipação, o que certamente diminuiria essa temperatura.

De qualquer forma, pode-se ter uma breve intuição da contribuição dos coolers com este problema em 2D. Estima-se que cada cooler consuma de 1 a 3W de potência do alimentador, e para o processador, o cooler deve ser capaz de dissipar o calor gerado pelo mesmo em seu funcionamento normal. Ao olhar descrições de processadores, facilmente o valor do calor gerado pelo mesmo em Watts é encontrado especificado como TDP (sigla para *Thermal Design Power*), entretanto, para os coolers, o parâmetro que nos é fornecido é o CFM (sigla para *Cubic Feet per Minute*), que é uma medida para a vazão de ar que os mesmos conseguem impor à máquina. Segundo o material fornecido pela *Sunon Technology*[8] a potência dissipada por um cooler para este projeto pode ficar estimada em 5W, sendo comum a coolers com vazão máxima em torno de 25CFM, que pe o caso da ventoinha especificada anteriormente(Cooler Amd/Intel Ice Edge Mini Fs V2. 0 Super Silent Deepcool, Super Silent Ice Edge Mini FS V2.0).

Então, foi aplicada a dissipação de 5W para os componentes com geração, e o resultado melhorou drásticamente no regime permanente:

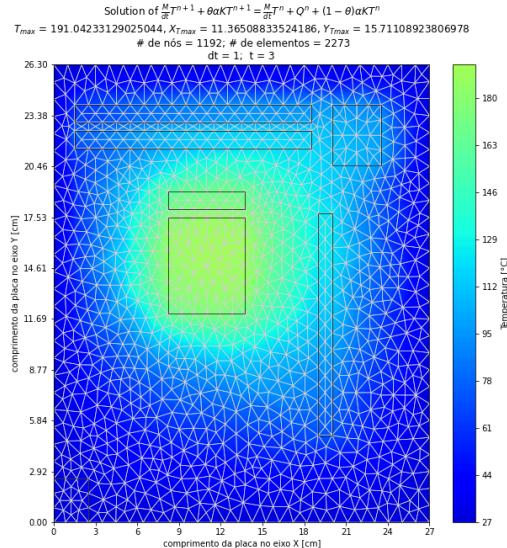


Figura 7: Resultado em regime permanente com utilização do Cooler

Ainda assim, 191°C ainda é uma temperatura muito alta para o processador trabalhar. Entretanto, a análise atual não leva em consideração

efeitos convectivos e torna-se muito crua para a representar a realidade. Mas de forma ilustrativa, pôde-se ver como a utilização de coolers torna-se necessária para este tipo de projeto e cabe ao projetista escolher estes itens de forma que melhor encaixe em seu projeto, sempre buscando obter o melhor custo-benefício.

3 Exercício 2 - Função Corrente

Uma outra possibilidade de utilização do método dos elementos finitos é a solução de problemas de mecânica dos fluidos. Muitas vezes é de interesse do projetista ou do cientista saber como determinado fluido irá escoar em um meio conhecido. Os motivos para isso são diversos, vide os exemplos abaixo:

- Estimar a produção de petróleo em um determinado poço, sabendo-se as condições de viscosidade e pressão do fluido no reservatório.
- Projetar tipos de *stent* para diferentes condições de circulação sanguínea no corpo
- Monitorar o funcionamento de dutos com fluidos viscosos, para evitar efeitos de acumulação ou efeitos químicos indesejáveis como a parafinização nos oleodutos.
- Projetar trocadores de calor que funcionem com diferentes geometrias, utilizando-se de efeitos convectivos e de dinâmica dos fluidos para obter uma maior eficiência
- entre outros.

Existem diversos projetos possíveis a partir do estudo da mecânica dos fluidos computacional, cabe à imaginação do projetista e à possibilidade física a realização destes grandes feitos de engenharia. Para o contexto atual, será solucionado um dos problemas mais simples em mecânica dos fluidos, que é o de escoamento entre placas planas e sem atrito.

3.1 A função corrente

Em escoamentos bidimensionais, a vorticidade pode ser facilmente obtida através da equação de Poisson:

$$\omega_z = \frac{\partial v_x}{\partial x} - \frac{\partial v_y}{\partial y} = \frac{\partial}{\partial x} \left(-\frac{\partial \psi}{\partial x} \right) - \frac{\partial}{\partial y} \left(\frac{\partial \psi}{\partial y} \right) \quad (7)$$

Desta forma, pode-se concluir que:

$$\nabla^2 \psi = -\omega_z$$

Porém, como o atrito não está sendo considerado, o escoamento torna-se irrotacional, e obtém-se então a equação de Laplace:

$$\nabla^2 \psi = 0$$

Para resolver utilizando o método dos elementos finitos então, basta ponderar a equação, multiplicando-a pela função peso w e integrar no domínio do problema. Após isso, utilizar a caracterização de Galerkin para obter a equação em sua forma fraca.

$$\int_{\Omega} w \nabla^2 \psi d\Omega = 0$$

Reduzindo então a ordem do termo da derivada segunda utilizando a identidade de Green e considerando que para as condições de contorno que serão utilizadas (Neumann e Dirichlet) a função peso no contorno é nula. Pode-se chegar ao seguinte termo:

$$\int_{\Omega} \nabla w \cdot \nabla \psi d\Omega = 0$$

Sendo então:

$$\psi(x, y) = \sum_{i=0}^{\infty} N_i(x, y) a_i; \quad w(x, y) = \sum_{j=0}^{\infty} N_j(x, y) b_j$$

Desta forma, a equação em sua forma final pode ser descrita da seguinte maneira:

$$\sum_{i=0}^n \left(\sum_{j=0}^n \int_{\Omega} \nabla N_i \cdot \nabla N_j d\Omega \right) a_i = 0$$

Este operador já é conhecido, e sabe-se que ele resultará na matriz de rigidez \mathbf{K} . Logo, a equação na forma matricial fica da seguinte maneira:

$$\mathbf{K}\psi = 0 + c.c \therefore \psi = \mathbf{K}^{-1}(0 + c.c.) \quad (8)$$

Resolvendo a equação da função corrente, pode utilizar-se das equações pertinentes para encontrar os campos de velocidades. Desta forma, os campos de velocidade podem ser entendidos como gradientes da função corrente (Imagine que a função corrente signifique exatamente como a velocidade varia no eixo X e em Y, se conseguir derivar parcialmente somente em um dos

eixos, consegue-se então o campo de velocidades no outro, e vice-versa), logo, utilizará-se as matrizes de gradiente G_x e G_y :

$$v_x = \mathbf{M}^{-1}(G_y\psi); \quad v_y = \mathbf{M}^{-1}(-G_x\psi) \quad (9)$$

Para a simulação, foram testadas malhas que representam uma distância entre placas de 1m e a região de controle se estende por 3m de comprimento. no centro da geometria foram colocados obstáculos para a observação do comportamento das velocidades dadas as diferentes geometrias.

3.2 Resultados da Simulação

O resultado para as placas paralelas sem obstáculos é bastante obsoleto e desinteressante, sendo apenas um gradiente para a função corrente e valores constantes próximos de 1 e zero para os campos de velocidade v_x e v_y , respectivamente. Sendo assim, aqui apresentam-se resultados obtidos ao simular outras geometrias, a primeira com a presença de um cilindro no meio, e a outra com a presença de um aerofólio, ambas as malhas sendo geradas no programa GMsh e o resultado exibido utilizando a biblioteca matplotlib, do Python.

Para a geometria com o cilindro, os resultados foram os seguintes:

- Função corrente ψ :

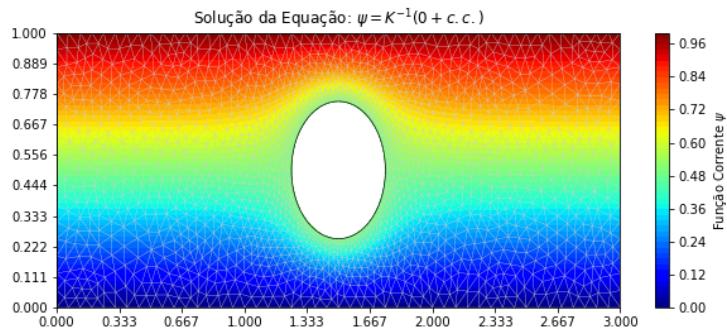


Figura 8: Função Corrente para a malha com um cilindro entre as paredes

- Campo de velocidades v_x :

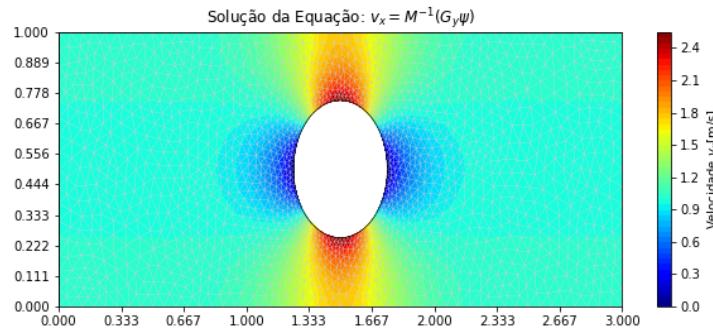


Figura 9: Campo de velocidades em x para a malha com um cilindro entre as paredes

- Campo de velocidades v_y :

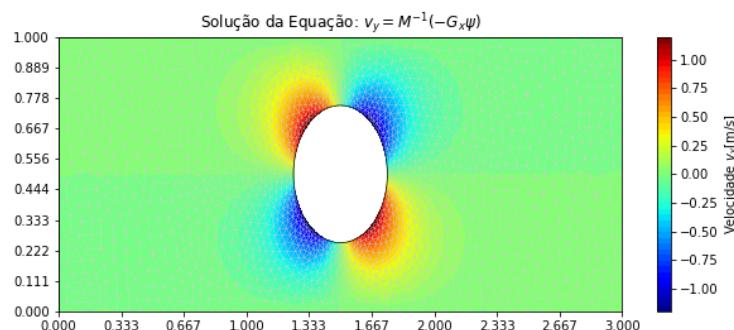


Figura 10: Campo de velocidades em y para a malha com um cilindro entre as paredes

Já para a geometria com o aerofólio, diferentes resultados puderam ser observados, como mostrado a seguir:

- Função corrente ψ :

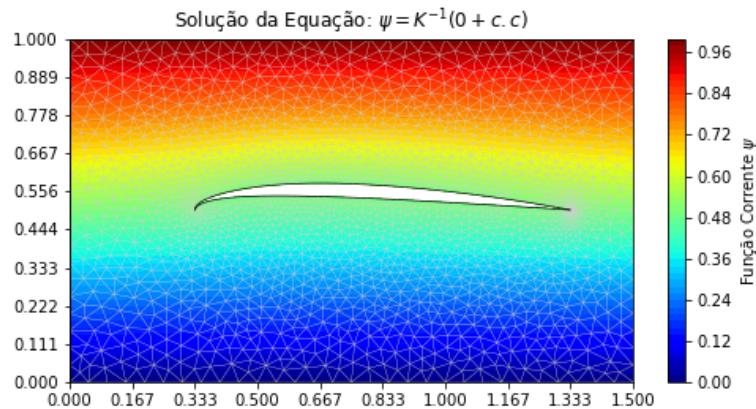


Figura 11: Função Corrente para a malha com um aerofólio entre as paredes

- Campo de velocidades v_x :

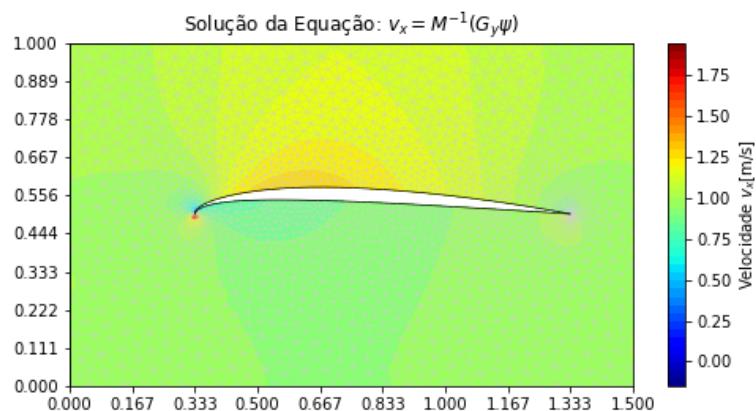


Figura 12: Campo de velocidades em x para a malha com um aerofólio entre as paredes

- Campo de velocidades v_y :

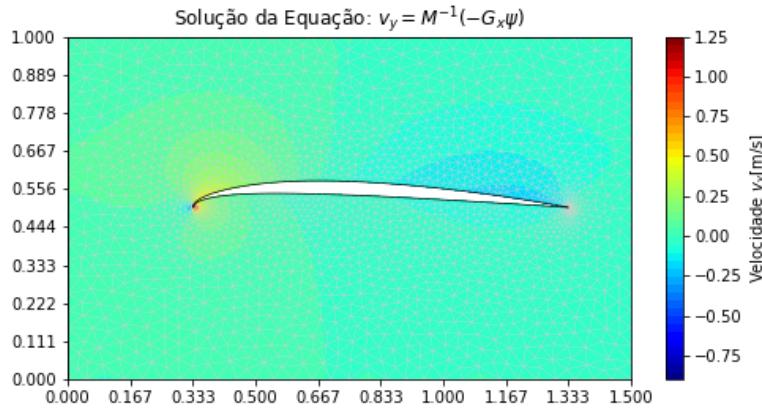


Figura 13: Campo de velocidades em y para a malha com um aerofólio entre as paredes

Os códigos para o processamento das malhas e simulações podem ser encontrados no apêndice deste documento, ou caso o leitor se sinta mais à vontade com o gerenciador de versões *Git*, basta seguir para este repositório no Github.

4 Exercício 3 - Função Corrente-Vorticidade

Para resolver a equação da corrente considerando a vorticidade, tem-se então os campos de corrente e velocidade variando com o tempo, conforme introduz-se os valores para a vorticidade no contorno.

Através da hipótese de escoamentos com fluidos newtonianos e incompressíveis, a equação da quantidade de movimento pode ser caracterizada como a equação de **Navier-Stokes**, conforme mostrada abaixo:

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = -\frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{v} + \mathbf{g} \quad (10)$$

Porém, através de identidades vetoriais e da definição de potencial de velocidades, é possível reescrever esta equação. A hipótese a ser assumida é a de conservação de massa no sistema, sendo assim:

$$\nabla \cdot \mathbf{v} = 0$$

As demais manipulações algébricas podem ser visualizadas no material do professor Gustavo Rabello[4]. Entretanto, a forma forte final (escalar) da equação do transporte de vorticidade bidimensional pode ser escrita desta maneira:

$$\frac{\partial w_z}{\partial t} + \mathbf{v} \cdot \nabla w_z = \nu \nabla^2 w_z + \nabla x \mathbf{g} \quad (11)$$

Da mesma forma, é possível obter a função corrente através da vorticidade para escoamento de fluido com atrito. Para isso, basta se ater à definição de vorticidade:

$$\nabla^2 \psi = -\omega_z \quad (12)$$

O termo de gravidade pode ser desacoplado da equação ao utilizar a aproximação de Boussinesq, que acopla efeitos gravitacionais em fluido com diferentes temperaturas através da redefinição de ρ .

Tem-se então o seguinte conjunto de equações:

$$\begin{aligned} \frac{\partial w_z}{\partial t} + \mathbf{v} \cdot \nabla w_z &= \nu \nabla^2 w_z \\ \nabla^2 \psi &= -\omega_z \end{aligned}$$

As condições de contorno para ψ são fixas no tempo, enquanto as da vorticidade ω_z são variáveis a cada iteração. Realizando o mesmo procedimento explicado anteriormente para a obtenção da equação na forma fraca pelo método dos elementos finitos, obtém-se a seguinte equação:

$$\mathbf{M} \left(\frac{\omega_z^{n+1} - \omega_z^n}{\Delta t} \right) + \mathbf{v} \cdot \mathbf{G} + \nu \mathbf{K} \omega = 0 \quad (13)$$

Para resolver o produto escalar $\mathbf{v} \cdot \mathbf{G}$, é necessário desacoplar o gradiente nos eixos ortogonais e multiplicar as componentes de velocidade pela matriz de identidade, da seguinte forma:

$$\mathbf{v} \cdot \mathbf{G} = v_x \mathbf{I} G_x + v_y \mathbf{I} G_y$$

No caso da equação da temperatura, ficou a dúvida se o termo com a matriz de rigidez seria explícito ou implícito. Para a solução da função de corrente-vorticidade isso não será mais uma dúvida, o termo com a matriz de rigidez \mathbf{K} deve permanecer implícito, ao passo que o termo que pode ou não ser implícito é o produto escalar $\mathbf{v} \cdot \mathbf{G}$. Isto ocorre pois no caso desta equação, a extrapolação ao explicitar \mathbf{K} torna-se muito grande, é um problema bem mais sensível ao passo de tempo.

Por fim, após aplicar a solução de ω_z para o contorno, é desejável obter a função corrente a partir da relação explicitada na equação (12). A mesma equação pode ser discretizada através do método dos elementos finitos. Dessa forma, sua forma fraca pode ser observada abaixo:

$$\mathbf{K}\psi = \mathbf{M}\omega_z \quad (14)$$

Sendo assim, para encontrar ψ através do algoritmo sabendo os valores de ω_z , basta fazer:

$$\psi = K^{-1}\mathbf{M}\omega_z$$

Através das relações equacionadas em (9), pode-se obter os campos de velocidade v_x e v_y , uma vez que a função corrente foi desvendada.

Esse tipo de simulação é mais complexa e sensível, então é bastante importante se manter atento aos parâmetros utilizados como número de Reynolds e passo de tempo. Se for necessário, ajustar para números menores, de forma que o problema seja corretamente solucionado pelo algoritmo.

4.1 Resultados da simulação

Foram utilizados 3 tipos de geometria e condições de contorno diferentes para a observação do fenômeno do atrito nas paredes, a primeira consistiu num pequeno cilindro funcionando como barreira em um escoamento entre placas planas. O segundo caso foi um problema cuja solução foi deveras mais sensível, apesar das condições de contorno serem relativamente fáceis, é o problema da cavidade. O terceiro caso é um problema conhecido em mecânica dos fluidos, que basicamente é quando se tem um canal e há uma abertura brusca no decorrer do mesmo. Os resultados e as condições de contorno aplicadas podem ser mostrados abaixo:

4.1.1 Pequeno cilindro entre paredes planas

Para este caso, foram utilizadas condições de contorno de Dirichlet para ω_z iterando no tempo, ao passo que as condições de contorno para ψ foram de os valores obedecerem uma função linear na entrada, ao passo que nas paredes o valor fosse fixo. Na saída foi aplicada a condição de Neumann ($\nabla\psi \cdot \mathbf{n} = 0$). Para as velocidades, as condições foram que nas paredes elas fossem zeradas, ao passo que na entrada a componente x da mesma fosse considerada constante. Sendo assim, obteve-se o seguinte resultado:

- Animação (Clique para visualizar):

Reiterando: É aconselhável que o leitor abra este arquivo no *Adobe Acrobat Reader DC*, para desfrutar dos conteúdos animados.

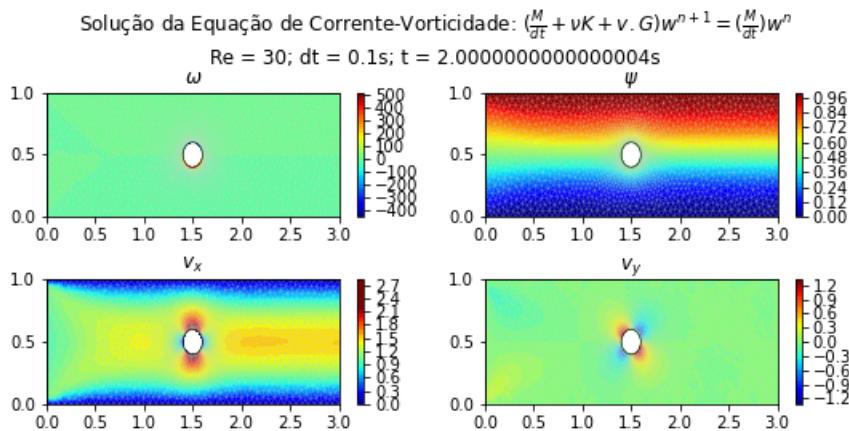


Figura 14: Estado final da simulação para pequeno cilindro entre paredes planas

4.1.2 Escoamento numa cavidade

Para este caso, também foram utilizadas condições de contorno de Dirichlet para ω_z iterando no tempo, ao passo que as condições para ψ foram zeradas em todo o contorno. Para as velocidades, as condições foram que em todo o contorno elas fossem zeradas, com exceção do topo, onde v_x é constante. Sendo assim, obteve-se o seguinte resultado:

- Animação (Clique para visualizar):

Após 20 iterações, pode então ser observada:

Solução da Equação de Corrente-Vorticidade: $(\frac{\partial}{\partial t} + vK + v \cdot G)w^{n+1} = (\frac{\partial}{\partial t})w$
 $Re = 10; dt = 0.01s; t = 0.2000000000000004s$

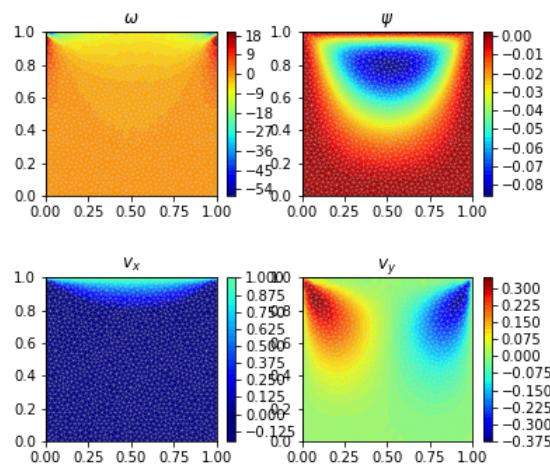


Figura 15: Estado final da simulação para uma pequena cavidade

4.1.3 Escoamento no canal com abertura

Para o último caso, as condições de contorno utilizadas foram as mesmas utilizadas no primeiro caso, com o diferencial da alteração na geometria das paredes. Obteve-se então:

- Animação (Clique para visualizar):

Foram utilizadas 40 iterações, com Reynolds 30 e passo de tempo de 0,05 segundo:

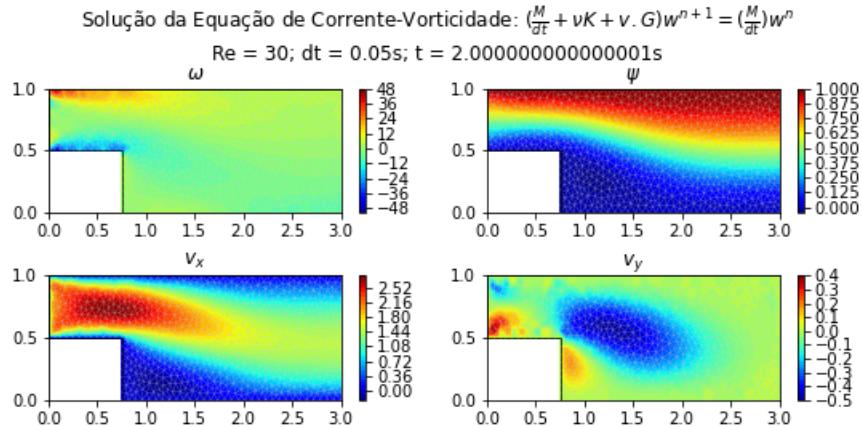


Figura 16: Estado final da simulação para o escoamento em degrau

5 Conclusão

O método dos elementos finitos mostra-se então um grande aliado à vida dos engenheiros de todo o planeta, tornando possível então a solução de problemas que não são possíveis de se resolver analíticamente e obtendo resultados favoráveis para o progresso da ciência. É importante reiterar que este não é o único método a ser utilizado, sendo possível resolver problemas por métodos mais simples, como o de diferenças finitas, ou outros métodos que podem ser bem mais complexos, como o de volumes finitos e a utilização de redes neurais fisicamente informadas, as PINNs. Ainda assim, é possível perceber que há uma infinidade de aspectos a se melhorar em diversos métodos, e para isso, é necessário que engenheiros e entusiastas coloquem a mão na massa para construir ciência e contribuir para uma sociedade com melhores condições de resolver seus problemas.

Agora, como nota pessoal, deixo aqui meu agradecimento ao professor Gustavo Rabello, pelo excelente curso que foi ministrado. Eu já possuía interesse pelo tema do método dos elementos finitos, mas seu entusiasmo e boa didática ao ensinar me fizeram ficar fascinado com o assunto, a ponto de me dedicar horas a fio para a melhor compreensão dos algoritmos. Gostaria de agradecer também aos meus colegas de turma, que foram essenciais nas horas de dúvidas e que trabalharam junto comigo fazendo código.

6 Referências

- [1] ANJOS, G.R. COMPUTAÇÃO CIENTÍFICA PARA ENGENHEIROS - Universidade Federal do Rio de Janeiro - UFRJ. Apostila do curso de Transferência de Calor e Mecânica dos Fluidos Computacional. 2020. 101 pg.
- [2] SILVA, J.A. – METODO DE ELEMENTOS FINITOS EM PYTHON COM A ABORDAGEM LAGRANGIANA EULERIANA PARA AS OSCILAÇÕES DE UM CILINDRO EM UM ESCOAMENTO TRANSVERSAL - Universidade Federal do Rio de Janeiro. Projeto Final. pg. 101. Novembro de 2020;
- [3] Autor Desconhecido – Introdução ao Método dos Elementos Finitos - Universidade Federal do Paraná. Apostila. pg. 93;
- [4] ANJOS, G.R. Projeto Final - Universidade Federal do Rio de Janeiro - UFRJ. Folha de Apoio para Projeto Final na disciplina Transferência de Calor e Mecânica dos Fluidos Computacional. pg. 43.
- [5] Descubra quanta energia seu PC consome e como reduzir o consumo - Canaltech. Matéria informativa sobre consumo energético dos computadores. <https://canaltech.com.br/desktop/descubra-quanta-energia-seu-pc-consome-e-como-reduzir-isso/>
- [6] OZISIK, M.N. - TRANSFERÊNCIA DE CALOR, Um Texto Básico. 1999. Editora Guanabara Koogan S.A. pg. 661
- [7] KINAST, Priscila - Qual temperatura máxima chega um processador? - Matéria informativa sobre as temperaturas usuais em processadores no site Oficina da Net. Publicada 2 de Abril de 2019. <https://www.oficinadanet.com.br/hardware/25291-qual-temperatura-maxima-chega-um-processador>
- [8] How to Select the Right Fan or Blower - Sunon Technology. Material com instruções para escolha de ventoinhas. 6pg.

A Apêndice - Códigos

A.1 Simulação do problema de transferencia de calor transiente

```
def montaKM(X,Y,IEN,regions):
    npoints = X.shape[0]
    q = 65
    cooler = 1
    if(cooler == 1):
        q -= 33.4
    cpSi = .867 ##
    rhoSi = 2.659
    kappa_Si = 1.61
    alpha_SiAl = kappa_Si/(rhoSi*cpSi)#(7.172*1e-5)
    cpPet = .712 ##
        https://www.plastmetal.com.br/tabelas/820116434bbd/tabela\_de\_propriedades\_polietileno\_sintetico.html
    rhoPet = 1.790
    kappa_Pet = 0.22
    alpha_ladeVidro = kappa_Pet/(cpPet*rhoPet)#(22.6*1e-7) #
        https://www.braskem.com.br/Portal/Principal/Arquivos/html/boletim\_tecnico/Tabela\_de\_propriedades\_da\_materia\_prima.html
    cpCu = .383
    rhoCu = 8.960
    kappa_Cu = 3.86
    alpha_Cu = kappa_Cu/(rhoCu*cpCu)#(11.234*1e-5)
    cpAl = .921
    rhoAl = 2.700
    kappa_Al = 2.04
    alpha_Al = kappa_Al/(rhoAl*cpAl)#(8.418*1e-5)
    alpha = np.ones(len(IEN), dtype='float')
    kappa = np.ones(len(IEN), dtype='float')
    rho = np.ones(len(IEN), dtype='float')
    cv = np.ones(len(IEN), dtype='float')
    Q = 0*np.ones((npoints),dtype='float')
    K = np.zeros( (npoints,npoints),dtype='float' )
    M = np.zeros( (npoints,npoints),dtype='float' )
    for elem in range(1,len(IEN)):
        [v1,v2,v3] = IEN[elem]
        xi,yi = X[v1],Y[v1]
        xj,yj = X[v2],Y[v2]
        xk,yk = X[v3],Y[v3]
```

```

ai = xj*yk - xk*yj
aj = xk*yi - xi*yk
ak = xi*yj - xj*yj
bi = yj - yk
bj = yk - yi
bk = yi - yj
ci = xk - xj
ck = xi - xk
cj = xj - xi
if(regions[elem] == 1): ##Base da Placa
    kappa[elem] = kappa_Pet
    cv[elem] = cpPet
    rho[elem] = rhoPet
    alpha[elem] = alpha_ladeVidro
if(regions[elem] == 2): ## Alimentador
    kappa[elem] = kappa_Al
    cv[elem] = cpAl
    rho[elem] = rhoAl
    alpha[elem] = alpha_Al
    Q[v1] = q*0.1/(rho[elem]*cv[elem])
    Q[v2] = q*0.1/(rho[elem]*cv[elem])
    Q[v3] = q*0.1/(rho[elem]*cv[elem])
if(regions[elem] in [3,4]): ## Processador AMD4
    kappa[elem] = kappa_Si
    cv[elem] = cpSi
    rho[elem] = rhoSi
    alpha[elem] = alpha_SiAl
    Q[v1] = q*0.15/(rho[elem]*cv[elem])
    Q[v2] = q*0.15/(rho[elem]*cv[elem])
    Q[v3] = q*0.15/(rho[elem]*cv[elem])
if(regions[elem] == 5): ## SSD M2
    kappa[elem] = kappa_Cu
    cv[elem] = cpCu
    rho[elem] = rhoCu
    alpha[elem] = alpha_Cu
    Q[v1] = q*0.1/(rho[elem]*cv[elem])
    Q[v2] = q*0.1/(rho[elem]*cv[elem])
    Q[v3] = q*0.1/(rho[elem]*cv[elem])
if(regions[elem] in [6,7]): ## Memoria RAM DDR4
    kappa[elem] = kappa_Cu
    cv[elem] = cpCu
    rho[elem] = rhoCu
    alpha[elem] = alpha_Cu

```

```

Q[v1] = q*0.1/(rho[elem]*cv[elem])
Q[v2] = q*0.1/(rho[elem]*cv[elem])
Q[v3] = q*0.1/(rho[elem]*cv[elem])
if(regions[elem] == 8): ## Placa de Video SATA 6Gbs
    kappa[elem] = kappa_Si
    cv[elem] = cpSi
    rho[elem] = rhoSi
    alpha[elem] = alpha_SiAl
    Q[v1] = q*0.15/(rho[elem]*cv[elem])
    Q[v2] = q*0.15/(rho[elem]*cv[elem])
    Q[v3] = q*0.15/(rho[elem]*cv[elem])
A = np.absolute((1/2)*np.linalg.det([[1,xi,yi],
                                      [1,xj,yj],
                                      [1,xk,yk]]))
ke_x = (1/(4*A))*np.array([[bi**2,bi*bj,bi*bk],
                             [bj*bi,bj**2,bj*bk],
                             [bk*bi,bk*bj,bk**2]])
ke_y = (1/(4*A))*np.array([[ci**2,ci*cj,ci*ck],
                             [cj*ci,cj**2,cj*ck],
                             [ck*ci,ck*cj,ck**2]])
ke = ke_x + ke_y
me = (A/12)*np.array([[2,1,1],
                       [1,2,1],
                       [1,1,2]])
for i_loc in range(0,3):
    i_glb = IEN[elem,i_loc]
    for j_loc in range(0,3):
        j_glb = IEN[elem,j_loc]

        K[i_glb,j_glb] += alpha[elem]*ke[i_loc,j_loc]
        M[i_glb,j_glb] += me[i_loc,j_loc]
return K,M,Q,npoly

```

```

def solveWithTheta(theta = 1,dt=0.5,lim_e=1e-5):
    """
    Equation
    -----
    (M/dt)*T+theta*alpha*K*T = (M/dt)*T0 + Q - (1-theta)*alpha*K*T0

```

```

Parameters
-----
theta : float, optional

```



```

    b = (M@Q)
for i in contorno_dell:
    A[i,:] = 0.0
    A[i,i] = 1.0
    b[i] = Tc
Ainv = np.linalg.inv(A)
T = Ainv@b
Z = T
xy = np.stack((X, Y), axis=-1)
verts = xy[IEN]
verts2 = xy[cc]
indice, = np.where(T == T.max())
XTmax = X[indice]
YTmax = Y[indice]
Tmax = T[indice]
t += dt
fig, ax = plt.subplots()
surf = ax.tricontourf(X,Y,Z,200,cmap=matplotlib.cm.jet,
                      extent=(X.min(),
                               X.max(), Y.min(), Y.max()),vmin=0,vmax=350)
fig.set_size_inches(10, 10)
ax=plt.gca()
pc =
    matplotlib.collections.PolyCollection(verts,edgecolors=('lightgray',),
                                           facecolor='None',
                                           linewidths=(0.7,))
ax.add_collection(pc)
pc2 =
    matplotlib.collections.PolyCollection(verts2,edgecolors=('black',),
                                           facecolor='None',
                                           linewidths=(0.7,))
ax.add_collection(pc2)
plt.xlabel('comprimento da placa no eixo X [cm]')
plt.ylabel('comprimento da placa no eixo Y [cm]')
#surf = plt.imshow(X,Y,Z, interpolation='quadric',
#                  origin='lower',
cbar = plt.colorbar(surf,shrink=1.0, aspect=20)
cbar.set_label('Temperatura')
labx = np.linspace(X.min(),X.max(),nx)
laby = np.linspace(Y.min(),Y.max(),ny)
plt.xticks(labx)
plt.yticks(laby)
subname = '????'

```

```

if(theta == 0):
    subname = 'resultados_explicito_placa_dell'
if(theta == 1):
    subname = 'resultados_implicito_placa_dell_v6'
if(theta == 1/2):
    subname = 'resultados_crank_nicholson_placa_dell'
plt.savefig(os.path.join(name, subname, f'{index:04}.png'))
plt.show()
index += 1
## mean squared error MSE
e = np.sum((T - Tpast)**2)/T.shape[0]
fp_in = f"./{name}/{subname}/*.png"
fp_out = f"./{name}/{subname}/simulacao.gif"

img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
         save_all=True, duration=dt*1000, loop=0)

```

A.2 Simulação do problema da função corrente

As partes de plot serão omitidas para não ficar repetitivo, caso queira olhar mais a fundo, visite o Github:

```

def solveCorrente():
    c2 = 1
    c1 = 0
    v = .1
    msh = meshio.read('aerofolio.msh')
    X = msh.points[:,0]
    Y = msh.points[:,1]
    IEN = msh.cells['triangle']
    IENbound = msh.cells['line']
    IENboundTypeElem = list(msh.cell_data['line'][gmsh:physical]
                           - 1)
    boundNames = list(msh.field_data.keys())
    IENboundElem = [boundNames[elem] for elem in IENboundTypeElem]
    npoints = len(X)
    ne = IEN.shape[0]

    # cria lista de nos do contorno
    cc = np.unique(IENbound.reshape(IENbound.size))
    ccName = [[] for i in range( npoints )]

```

```

for elem in range(0,len(IENbound)):
    ccName[ IENbound[elem][0] ] = IENboundElem[elem]
    ccName[ IENbound[elem][1] ] = IENboundElem[elem]
regions = msh.cell_data['triangle']['gmsh:geometrical']

vx = np.zeros(npoints,dtype='float64')
vy = np.zeros((npoints),dtype='float64')
K,M,Gx,Gy = montaKM(X,Y,IEN,regions)
index = 0
Minv = np.linalg.inv(M)
A = K.copy()
b = np.zeros( (npoints),dtype='float' )
bc = 0*np.ones( (npoints),dtype='float' )
for i in cc:
    A[i,:] = 0.0
    A[i,i] = 1.0
    if ccName[i] == 'top':
        bc[i] = c2
        b[i] = bc[i]
    if ccName[i] == 'bottom':
        b[i] = c1
    if ccName[i] == 'left':
        b[i] = Y[i]
    if ccName[i] == 'hole':
        b[i] = (max(Y)-min(Y))/2
## Solucao vorticidade
Ainv = np.linalg.inv(A)
# b -= bc
psi = Ainv@b
## Campo de velocidades
b_3 = np.dot(Gy,psi)
M3 = M.copy()
vx = np.linalg.solve(M3,b_3)

b_4 = np.dot(Gx,psi)
M4 = M.copy()
vy = -np.linalg.solve(M4,b_4)

```

A.3 Simulação do problema da função corrente-vorticidade

```

def solveVortexWithTheta(theta = 1,dt=0.1,lim_e=1e-5):
    nx = 10

```

```

ny = 10
t = 0
Re = 30
nu = 1/Re
e=1.
msh = meshio.read('duto-furo-menor.msh')
X = msh.points[:,0]
Y = msh.points[:,1]
IEN = msh.cells['triangle']
cc = msh.cells['line']
npoints = len(X)
ne = IEN.shape[0]
regions = msh.cell_data['triangle']['gmsh:geometrical']
parede_top =
    [1,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81]
parede_bot =
    [3,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119]
entrada = [5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
saida = [67,68,69,70,71,72,73,74,75,76,77,78,79,80,81]
furo = [4]

for i in range(129,192):
    furo.append(i)
contorno = [*entrada,*parede_top,*parede_bot,*saida,*furo]
Yfuro = Y[furo]
centro = (Yfuro.max() + Yfuro.min())/2
# contorno,entrada,parede_top,saida,parede_bot =
    getContorno(IENBound)
w = np.zeros((npoints),dtype='float64')
vx = np.zeros((npoints),dtype='float64')
vy = np.zeros((npoints),dtype='float64')

for i in parede_top:
    vx[i] = 0.0
    vy[i] = 0.0
for i in parede_bot:
    vx[i] = 0.0
    vy[i] = 0.0
for i in entrada:
    vx[i] = 1.0
    vy[i] = 0.0
for i in furo:
    vx[i] = 0.0

```

```

vy[i] = 0.0
psi = np.zeros((npoints), dtype='float64')
K,M,Gx,Gy = montaKMG(X,Y,IEN)
index = 0
w = np.linalg.solve(M, (Gx@vy - Gy@vx))
while(index <= 139):
    t += dt
    Minv = np.linalg.inv(M)
    gv = Gx@vy-Gy@vx
    omegacc = np.linalg.solve(M, (Gx@vy - Gy@vx))

    I = np.identity(npoints)
    vxI = vx*I
    vyI = vy*I

    VGO = vxI@Gx + vyI@Gy
    Kest = montaKest(X,Y,IEN,vx,vy,dt)

    A_w = M.copy()/dt + nu*K.copy() + ((vx*I)@Gx + (vy*I)@Gy)
    b_w = M.copy()/dt @ w
    for i in contorno:
        A_w[i,:] = 0.0
        A_w[i:i] = 1.0
        b_w[i] = omegacc[i]
    ## Solucao vorticidade
    w = np.linalg.solve(A_w,b_w)
    ## Solucao funcao corrente
    A_psi = K.copy()
    b_psi = M@w
    for i in parede_top:
        A_psi[i,:] = 0.0
        A_psi[i,i] = 1.0
        b_psi[i] = 1.0
    for i in parede_bot:
        A_psi[i,:] = 0.0
        A_psi[i,i] = 1.0
        b_psi[i] = 0.0
    for i in entrada:
        A_psi[i,:] = 0.0
        A_psi[i,i] = 1.0
        b_psi[i] = Y[i]
    for i in furo:
        A_psi[i,:] = 0.0

```

```
A_psi[i,i] = 1.0
b_psi[i] = centro
A_psiinv = np.linalg.inv(A_psi)
psi = A_psiinv@b_psi
## Campo de velocidades
Gypsi = Gy@psi
vx = Minv@Gypsi
Gxpsi = -Gx@psi
vy = Minv@Gxpsi
## resolvendo contornos
for i in parede_top:
    vx[i] = 0.0
    vy[i] = 0.0
for i in parede_bot:
    vx[i] = 0.0
    vy[i] = 0.0
for i in entrada:
    vx[i] = 1.0
    vy[i] = 0.0
for i in furo:
    vx[i] = 0.0
    vy[i] = 0.0
```
