Aluno: Raphael Henrique Braga Leivas
Matrícula: 2020028101
Professor Responsável: Márcio Ziviani

Código fonte LaTeX desse arquivo pode ser visto em meu GitHub pessoal:
https://github.com/RaphaelLeivas/latex/tree/main/TermoComp

1 Problema

Precisamos criar um algoritmo para determinar a temperatura de equilíbrio da superfície da face esquerda T_e de uma placa vertical. Temos as seguintes informações dadas:

- Condutividade térmica da placa: $k = 2,5 \text{ W/m}^2\text{K}$
- Comprimento: $W = 5,0 \text{ m}$
- Altura: $H = 2,0 \text{ m}$
- Espessura: $L = 0,25 \text{ m}$
- Velocidade do ar ambiente na face esquerda: $u_\infty = 3 \text{ m/s}$
- Temperatura do ar ambiente na face esquerda: $T_\infty = 300 \text{ K}$
- Fluxo de calor prescrito sobre a face esquerda: $q_p^{''} = 750 \text{ W/m}^2$
- Temperatura da face direita: $T_d = 350 \text{ K}$

2 Solução

Nota: O código completo desenvolvido na solução pode ser visto no final desse arquivo, em Anexo.

De posse dessas informações, fazemos um desenho esquemático do problema, exibido na Figura 2.1. Como mostra a Figura 2.1, traçamos uma superfície de controle na face esquerda. Aplicando a primeira lei da termodinâmica sobre essa superfície, temos

$$\dot{E}_{in} + \dot{E}_g - \dot{E}_{out} = \dot{E}_{st} \tag{2.1}$$

Como temos uma superfície de controle, não há energia armazenada e nem energia gerada. Além disso, no modelo da Figura 2.1, repare que todos os fluxos de calor estão entrando na superfície, de tal maneira que $\dot{E}_{out} = 0$. Assim, (2.1) se reduz a

$$\dot{E}_{in} = 0 \tag{2.2}$$

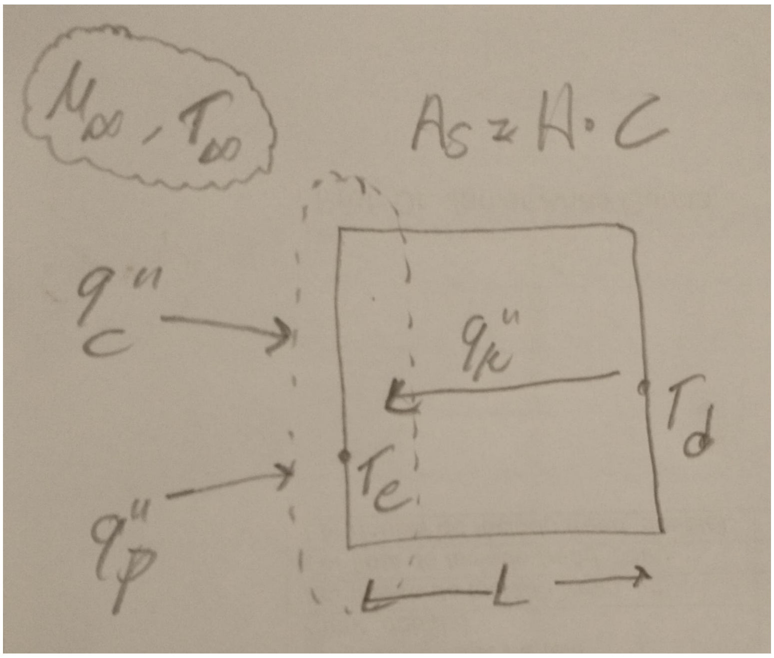
Expandindo o termo $\dot{E}_{in}$,

$$q_p^{''} + q_c^{''} + q_k^{''} = 0$$

$$q_p^{''} + h_c\left(T_\infty - T_e\right) + \frac{k}{L}\left(T_d - T_e\right) = 0$$

Isolando o coeficiente convectivo h_c, temos

Figura 2.1: Diagrama do problema posto, com superfície de controle destacada.



Fonte: elaboração própria.

$$h_c\left(T_\infty - T_e\right) = -q_p^{''} - \frac{k}{L}\left(T_d - T_e\right)$$

$$h_c = -\frac{q_p^{''} + \frac{k}{L}\left(T_d - T_e\right)}{T_\infty - T_e} \tag{2.3}$$

Como a velocidade do ar ambiente na face da placa é $u_\infty = 3\ \mathrm{m/s}$, temos que isso equivale a $10,8\ \mathrm{km/h}$. Para ter uma noção do quão rápido essa velocidade é, usamos a Tabela 2.1.

Tabela 2.1: Relação entre a velocidade do vento e seu efeito.

Nº de Beaufort	Descrição	Velocidade do vento (km/h)
0	Calmo	< 1.6
1	Ar leve	1.6 a 4.8
2	Brisa leve	6.4 a 11.2

Fonte: Traduzido de National Oceanic and Atmospheric Administation (NOAA). Disponível em: https://www.weather.gov/pqr/wind.

Assim, segundo a Tabela 2.1, temos que $u_\infty = 10,8\ \mathrm{km/h}$ equivale a apenas uma brisa leve, e portanto podemos considerar a convecção como natural. Nota: tentamos considerar a convecção como forçada, mas encontramos erros, conforme elaborado na seção 3.

Uma vez feita a análise inicial do problema, agora criamos um código em linguagem R para calcular iterativamente a temperatura da face esquerda até ela convergir para para a temperatura de equilíbrio. O primeiro passo é adicionar os dados do problema no código

```
# dados do problema
k <- 2.5
W <- 5
H <- 2
```

```
L <- 0.25
u_inf <- 3
T_inf <- 300
q_pres <- 750
T_d <- 350
```

Em seguida, definimos a tabela com as propriedades termofísicas do ar conforme o livro do Incropera. A primeira coluna da variável `Properties_Table` representa a temperatura do fluido, e as demais colunas representam respectivamente as propriedades termofísicas de massa específica ρ, viscosidade dinâmica μ, viscosidade cinética v, condutividade térmica do fluido k_f e número de Prandtl Pr.

```
Properties_Table = matrix(
    c(
      100, 3.5562, 71.1 * 10^-7, 2.00 * 10^-6, 9.34 * 10^-3, 0.786,
      150, 2.3364, 103.4 * 10^-7, 4.426 * 10^-6, 13.8 * 10^-3, 0.758,
      200, 1.7458, 132.5 * 10^-7, 7.590 * 10^-6, 18.1 * 10^-3, 0.737,
      250, 1.3947, 159.6 * 10^-7, 11.44 * 10^-6, 22.3 * 10^-3, 0.720,
      300, 1.1614, 184.6 * 10^-7, 15.89 * 10^-6, 26.3 * 10^-3, 0.707,
      350, 0.9950, 208.2 * 10^-7, 20.92 * 10^-6, 30.0 * 10^-3, 0.700,
      400, 0.8711, 230.1 * 10^-7, 26.41 * 10^-6, 33.8 * 10^-3, 0.690,
      450, 0.7740, 250.7 * 10^-7, 32.39 * 10^-6, 37.3 * 10^-3, 0.686,
      500, 0.6964, 270.1 * 10^-7, 38.79 * 10^-6, 40.7 * 10^-3, 0.684,
      550, 0.6329, 288.4 * 10^-7, 45.57 * 10^-6, 43.9 * 10^-3, 0.683,
      600, 0.5804, 305.8 * 10^-7, 52.69 * 10^-6, 46.9 * 10^-3, 0.685
    ),
    ncol = 6,
    byrow = TRUE
)

colnames(Properties_Table) <- c('Tf', 'rho', 'mu', 'v', 'kf', 'Pr')
rownames(Properties_Table) <- seq(1, nrow(Properties_Table), 1)
Properties_Table <- as.table(Properties_Table)
```

Agora definimos uma variável `max_iterations` que limita o número máximo de iterações para que T_e converja, a fim de evitar loops infinitos em tempo de compilação. Além disso, definimos o valor inicial ("chute") para T_e, e o adicionamos em uma lista que salva os valores calculados de T_e em cada iteração. Consideramos o valor inicial de T_e como a temperatura ambiente $T_e = 25\ °\mathrm{C} = 298\ \mathrm{K}$.

```
max_iterations <- 100
T_e_calculated <- 298 # chute inicial: Te = 298K (ambiente)
T_e_list <- c(T_e_calculated)
```

Agora definimos um loop `for`, em que cada iteração se calcula um valor de T_e até que ele converja.

```
for (i in 1:max_iterations) {

}
```

Dentro de cada iteração do loop, inicializamos uma variável T_f, que é a temperatura do fluido em que olharemos na tabela, definida por

$$T_f = \frac{T_e + T_\infty}{2} \tag{2.4}$$

No código, (2.4) é executada por

```
Tf <- (T_e_calculated + T_inf) / 2
```

O próximo passo é percorrer a tabela das propriedades termofísicas, identificando as temperaturas

$T_{f_{min}}$ e $T_{f_{max}}$ entre as quais a temperatura T_f se encontra, isto é, a faixa $T_{f_{min}} \leq T_f \leq T_{f_{max}}$.

```
# iniciais
Tf_min <- Properties_Table[1, 'Tf']
Tf_min_index <- 1
Tf_max <- Properties_Table[nrow(Properties_Table), 'Tf']
Tf_max_index <- nrow(Properties_Table)

# procura na tabela alguem com esse valor de Tf
for (j in 1:nrow(Properties_Table)) {
  if (Properties_Table[j, 'Tf'] <= Tf) {
    Tf_min <- Properties_Table[j, 'Tf']
    Tf_min_index <- j
  }

  if (Properties_Table[j, 'Tf'] > Tf) {
    Tf_max <- Properties_Table[j, 'Tf']
    Tf_max_index <- j
    break
  }
}
```

Isso deve ser feito pois T_f pode não coincidir exatamente com os valores tabelados. Assim, usamos a variável `interpolation_ratio` para realizar a interpolação dos valores das propriedades, dependendo do quão próximo T_f está de $T_{f_{min}}$ ou $T_{f_{max}}$.

```
interpolation_ratio <- (Tf - Tf_min) / (Tf_max - Tf_min)
```

Com a razão de interpolação, calculamos as propriedades termofísicas necessárias através de interpolação com os dados da tabela.

```
rho_min <- Properties_Table[Tf_min_index, 'rho']
rho_max <- Properties_Table[Tf_max_index, 'rho']
rho <- rho_min + (rho_max - rho_min) * interpolation_ratio

mu_min <- Properties_Table[Tf_min_index, 'mu']
mu_max <- Properties_Table[Tf_max_index, 'mu']
mu <- mu_min + (mu_max - mu_min) * interpolation_ratio

v_min <- Properties_Table[Tf_min_index, 'v']
v_max <- Properties_Table[Tf_max_index, 'v']
v <- v_min + (v_max - v_min) * interpolation_ratio

kf_min <- Properties_Table[Tf_min_index, 'kf']
kf_max <- Properties_Table[Tf_max_index, 'kf']
kf <- kf_min + (kf_max - kf_min) * interpolation_ratio

Pr_min <- Properties_Table[Tf_min_index, 'Pr']
Pr_max <- Properties_Table[Tf_max_index, 'Pr']
Pr <- Pr_min + (Pr_max - Pr_min) * interpolation_ratio
```

Com as propriedades termofísicas calculadas, agora calculamos os coeficientes adimensionais

```
Re_critical <- 50000
Re <- u_inf * L * rho / mu
Gr <- 9.81 * (1/Tf) * (abs(T_e_calculated - T_inf) * L^3) / (v^2)
Ra <- Gr * Pr
```

```
Nu <- (0.825 + (0.387 * Ra^(1/6)) / (1 + (0.492/Pr)^(9/16))^(8/27))^2
```

Note que, para o cálculo do Número de Nusselt, usamos a expressão (2.5), que é a mesma para escoamentos laminar e turbulento em uma placa plana vertical.

$$Nu = \left\{ 0,825 + \frac{0,387 Ra^{1/6}}{\left[1 + (0,492/Pr)^{9/16}\right]^{8/27}} \right\}^2 \tag{2.5}$$

Encontramos o coeficiente convectivo h_c através de

$$h_c = \frac{Nu \cdot K_f}{L} \tag{2.6}$$

No código,

```
hc <- Nu * kf / L
```

Com h_c calculado, encontramos a temperatura da face esquerda através da expressão (2.3), isolando T_e, obtendo

$$h_c T_\infty - h_c T_e = -q_p^{''} - \frac{k}{L} T_d + \frac{k}{L} T_e$$

$$-T_e \left(h_c + \frac{k}{L} \right) = -h_c T_\infty - q_p^{''} - \frac{k}{L} T_d$$

$$T_e = \frac{h_c T_\infty + q_p^{''} + \frac{k}{L} T_d}{h_c + \frac{k}{L}} \tag{2.7}$$

No código, calculamos T_e através de (2.7) e o inserimos na lista de T_e calculados

```
T_e_calculated <- ((k/L) * T_d + q_pres + T_inf * hc) / (hc + (k/L))
T_e_list <- append(T_e_list, T_e_calculated)
```

Ao final da iteração, verificamos se o valor calculado está dentro da tolerância de $0,1\%$ definida. Caso esteja dentro dessa tolerância, já está convergindo e deve sair do loop com o `break`. Caso contrário, continua para a próxima iteração do loop, com `next`.

```
tolerance <- 0.001 # 0.1%
if (abs(T_e_list[i + 1] - T_e_list[i]) < tolerance * T_e_list[i]) {
    break
} else {
    next
}
```

Finalmente, ao final das iterações do loop, plotamos os valores de T_e armazenados na lista, para que possamos visualizar a convergência.

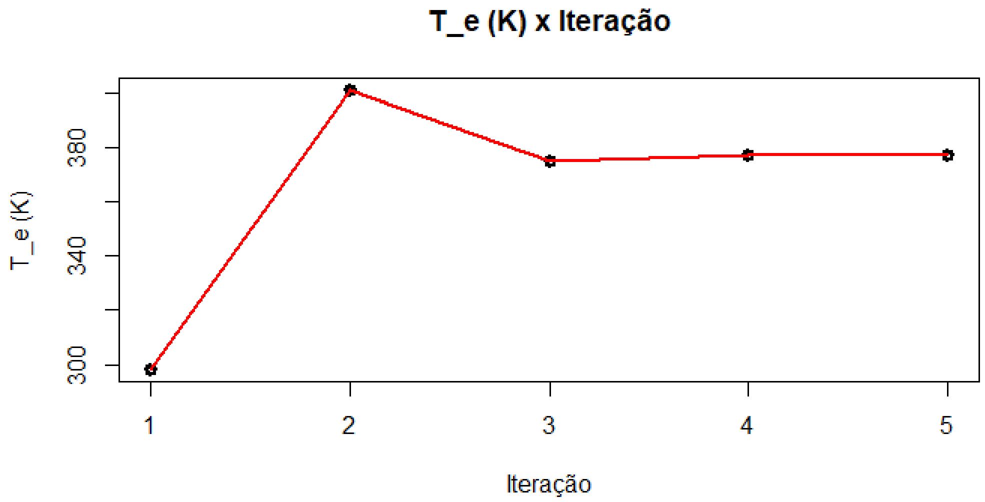
```
plot(
    T_e_list,
    main = "T_e (K) x Iteracao",
    xlab = "Iteracao",
    ylab = "T_e (K)",
    col = "black",
    lwd = 3
)

lines(T_e_list, col = "red", lwd = 2, lty = 1)
```

O gráfico plotado pelo software está exibido na Figura 2.2. A temperatura de equilíbrio para a superfície esquerda calculada pelo software foi de

$$\boxed{T_e = 377,19\ \mathrm{K}}$$

Figura 2.2: Evolução da temperatura T_e calculada pelo software a cada iteração do loop.

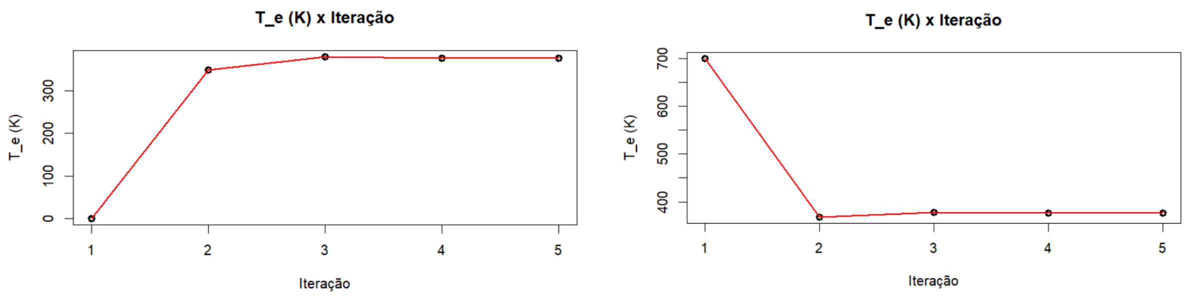


Fonte: elaboração própria.

3 Análise e Discussão dos Resultados

A robustez da solução pode ser verificada alterando o valor do "chute" inicial. Para valores iniciais de T_e extremos de $T_{e_{inicial}} = 0\ \mathrm{K}$ e $T_{e_{inicial}} = 700\ \mathrm{K}$, obtemos temperaturas de equilíbrio respectivamente de $T_e = 377,23\ \mathrm{K}$ e $T_e = 377,21\ \mathrm{K}$, comprovando que o valor de convergência do software não depende do chute inicial. As Figuras 3.1 (a) e (b) exibem respectivamente a convergência para cada valor extremo de T_e inicial usado.

Figura 3.1: Evolução da temperatura T_e calculada pelo software a cada iteração do loop, para T_e inicial (a) $0\ \mathrm{K}$ e (b) $700\ \mathrm{K}$.



(a) (b)

Fonte: elaboração própria.

Por fim, gostaríamos de elaborar um pouco mais sobre a decisão de considerar a convecção como natural, e não como forçada, apesar do ar sobre a placa possuir velocidade de $u_\infty = 10,8\ \mathrm{km/h}$. Se considerarmos a convecção como forçada, o regime de escoamento é determinado pelo Número de Reynolds Re, e o valor do Número de Nusselt Nu é dado por

$$Nu = \begin{cases} 0,664 Re^{1/2} Pr^{1/3}, & Re \leq 5 \cdot 10^5 \ \text{(Laminar)}, \\ \left(0,037 Re^{4/5} - 871\right) Pr^{1/3}, & Re > 5 \cdot 10^5 \ \text{(Turbulento)}, \end{cases} \tag{3.1}$$

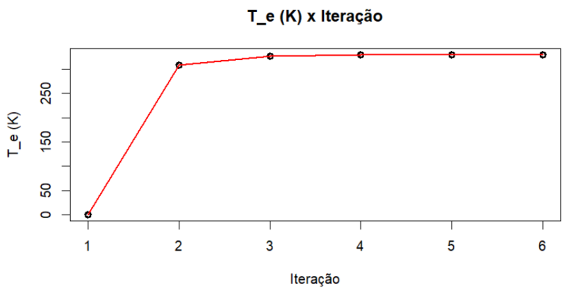
No código, a condição (3.1) pode ser feita através de

```
if (Re < Re_critical) {
    # escoamento laminar
    Nu <- 0.664 * (Re)^(1/2) * (Pr)^(1/3)
  } else {
    # escoamento turbulento
    Nu <- (0.037 * Re^(4/5) - 871) * (Pr)^(1/3)
  }
```

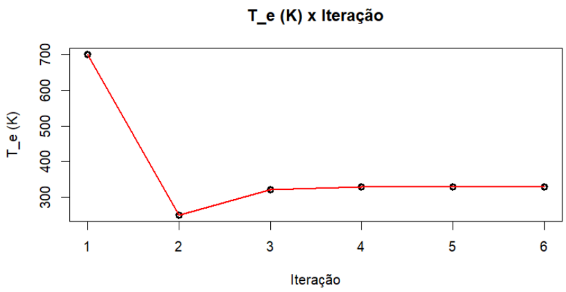
No entanto, verificamos que, em condição de escoamento turbulento, o Número de Nusselts calculado era negativo devido à parcela -871 da subtração, o que é claramente uma impossibilidade física. Assim, optamos por considerar a convecção como natural para evitar essa inconsistência.

Para melhor estudar o comportamento da solução considerando-se a convecção como forçada, vamos considerar a velocidade do fluido $u_\infty = 30$ m/s, o que equivale a $u_\infty = 108$ km/h, que claramente é convecção forçada. Nesse caso, compilamos o software novamente levando em consideração a condição (3.1), obtendo os gráficos da Figura 3.2.

Figura 3.2: Evolução da temperatura T_e calculada pelo software a cada iteração do loop, para T_e inicial (a) 0 K e (b) 700 K, com $u_\infty = 108$ km/h.



(a)



(b)

Fonte: elaboração própria.

As soluções usando convecção forçada para altos valores de u_∞ são robustas para vários valores inicias de T_e "chutados", como mostra a Figura 3.2, uma vez que ambas convergiram para $T_e = 329,68$ K. Portanto, é válido considerar a convecção como natural para baixos valores de u_∞ e forçada para altos valores de u_∞, uma vez que ambas convergem para a mesma temperatura de equilíbrio da face esquerda T_e para diferentes valores iniciais estimados no código.

4 Referências

INCROPERA, Frank, et. al. Fundamentals of Heat and Mass Transfer. 6 ed. John Wilhey & Sons Inc. 2007.

National Oceanic and Atmospheric Administation (NOAA). Estimating wind speed. Disponível em: https://www.weather.gov/pqr/wind. Acesso em 22 de abr. de 2023.

5 Anexo - Código completo em R desenvolvido

```
rm(list = ls())
# dev.off()

# dados do problema
k <- 2.5
W <- 5
H <- 2
L <- 0.25
u_inf <- 3
T_inf <- 300
q_pres <- 750
T_d <- 350

# propriedades termofisicas
Properties_Table = matrix(
  c(
    100, 3.5562, 71.1 * 10^-7, 2.00 * 10^-6, 9.34 * 10^-3, 0.786,
    150, 2.3364, 103.4 * 10^-7, 4.426 * 10^-6, 13.8 * 10^-3, 0.758,
    200, 1.7458, 132.5 * 10^-7, 7.590 * 10^-6, 18.1 * 10^-3, 0.737,
    250, 1.3947, 159.6 * 10^-7, 11.44 * 10^-6, 22.3 * 10^-3, 0.720,
    300, 1.1614, 184.6 * 10^-7, 15.89 * 10^-6, 26.3 * 10^-3, 0.707,
    350, 0.9950, 208.2 * 10^-7, 20.92 * 10^-6, 30.0 * 10^-3, 0.700,
    400, 0.8711, 230.1 * 10^-7, 26.41 * 10^-6, 33.8 * 10^-3, 0.690,
    450, 0.7740, 250.7 * 10^-7, 32.39 * 10^-6, 37.3 * 10^-3, 0.686,
    500, 0.6964, 270.1 * 10^-7, 38.79 * 10^-6, 40.7 * 10^-3, 0.684,
    550, 0.6329, 288.4 * 10^-7, 45.57 * 10^-6, 43.9 * 10^-3, 0.683,
    600, 0.5804, 305.8 * 10^-7, 52.69 * 10^-6, 46.9 * 10^-3, 0.685
  ),
  ncol = 6,
  byrow = TRUE
)

colnames(Properties_Table) <- c('Tf', 'rho', 'mu', 'v', 'kf', 'Pr')
rownames(Properties_Table) <- seq(1, nrow(Properties_Table), 1)
Properties_Table <- as.table(Properties_Table)

max_iterations <- 100
T_e_calculated <- 298 # chute inicial: Te = 298K (ambiente)
T_e_list <- c(T_e_calculated)

for (i in 1:max_iterations) {
  Tf <- (T_e_calculated + T_inf) / 2

  # iniciais
  Tf_min <- Properties_Table[1, 'Tf']
  Tf_min_index <- 1
  Tf_max <- Properties_Table[nrow(Properties_Table), 'Tf']
  Tf_max_index <- nrow(Properties_Table)

  # procura na tabela alguem com esse valor de Tf
  for (j in 1:nrow(Properties_Table)) {
    if (Properties_Table[j, 'Tf'] <= Tf) {
      Tf_min <- Properties_Table[j, 'Tf']
```

```
    Tf_min_index <- j
  }

  if (Properties_Table[j, 'Tf'] > Tf) {
    Tf_max <- Properties_Table[j, 'Tf']
    Tf_max_index <- j
    break
  }
}

# agora sabemos que Tf esta entre [Tf_min, Tf_max]
# pega a razao que diz o quao proximo esta de min ou max
interpolation_ratio <- (Tf - Tf_min) / (Tf_max - Tf_min)

# com a razao de interpolacao, acha as propriedades fisicas interpoladas
rho_min <- Properties_Table[Tf_min_index, 'rho']
rho_max <- Properties_Table[Tf_max_index, 'rho']
rho <- rho_min + (rho_max - rho_min) * interpolation_ratio

mu_min <- Properties_Table[Tf_min_index, 'mu']
mu_max <- Properties_Table[Tf_max_index, 'mu']
mu <- mu_min + (mu_max - mu_min) * interpolation_ratio

v_min <- Properties_Table[Tf_min_index, 'v']
v_max <- Properties_Table[Tf_max_index, 'v']
v <- v_min + (v_max - v_min) * interpolation_ratio

kf_min <- Properties_Table[Tf_min_index, 'kf']
kf_max <- Properties_Table[Tf_max_index, 'kf']
kf <- kf_min + (kf_max - kf_min) * interpolation_ratio

Pr_min <- Properties_Table[Tf_min_index, 'Pr']
Pr_max <- Properties_Table[Tf_max_index, 'Pr']
Pr <- Pr_min + (Pr_max - Pr_min) * interpolation_ratio

# rho <- 1.1614 # densidade do ar
# mu <- 184.6 * 10^-7 # viscosidade dinamica
# kf <- 26.3 * 10^-3 # condutividade termica do ar
# Pr <- 0.707 # numero de Prandlt

# calcula os adimensionais
Re_critical <- 50000
Re <- u_inf * L * rho / mu
Nu <- 0.0
Gr <- 9.81 * (1/Tf) * (abs(T_e_calculated - T_inf) * L^3) / (v^2)
Ra <- Gr * Pr

if (Re < Re_critical) {
  # escoamento laminar
  Nu <- 0.664 * (Re)^(1/2) * (Pr)^(1/3)
} else {
  # escoamento turbulento
  Nu <- (0.037 * Re^(4/5) - 871) * (Pr)^(1/3)
}
```

```
  # Nu <- (0.825 + (0.387 * Ra^(1/6)) / (1 + (0.492/Pr)^(9/16))^(8/27))^2

  # calculado o Numero de Nusselt, achamos o hc
  hc <- Nu * kf / L

  # para esse hc, o Te da 1 Lei e
  T_e_calculated <- ((k/L) * T_d + q_pres + T_inf * hc) / (hc + (k/L))
  T_e_list <- append(T_e_list, T_e_calculated)

  tolerance <- 0.001 # 0.1%

  # condicao de parada, tolerancia de 0.1% com o valor anterior
  if (abs(T_e_list[i + 1] - T_e_list[i]) < tolerance * T_e_list[i]) {
    break
  } else {
    next
  }
}

plot(
  T_e_list,
  main = "T_e (K) x Iteracao",
  xlab = "Iteracao",
  ylab = "T_e (K)",
  col = "black",
  lwd = 3
)

lines(T_e_list, col = "red", lwd = 2, lty = 1)
```