Trabalho 3:

Leonardo Maximo Silva, 200022172

Sumário

1 Introdução

O trabalho foi composto por uma Lista de 10 exercícios. Dos exercícios 1 a 5, a Equação de Condução do Calor foi resolvida para o caso unidimensional através do Método das Diferenças Finitas, sendo, posteriormente, comparada com a solução exata da equação para diferentes valores de tempo e de passo de tempo (dt). Para o exercício 5, utilizou-se do Método do Ghost-Point para se obter as condições de contorno adequadas para $\frac{\partial T}{\partial t} = 0$. Para os exercícios 6 a 8, a análise realizadas nos exercícios 1 a 5 foi repetida, porém, para o caso bidimensional da Equação da Condução do Calor. O exercício 9 consistiu da realização da Equação da Condução do Calor Bidimensional para o caso em que há um elemento (obstáculo) à temperatura constante na placa e, nos exercícios 10 e 11, resolveram-se Sistemas de Equações Lineares através dos métodos de Jacobi, Gauss-Seidel, SOR e Gradiente Conjugado (apenas no exercícico 11).

2 Exercício 1

Tomando o Método das Diferenças Finitas para aproximar a Equação da Condução do calor para o caso unidimensional 1,

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} \tag{1}$$

Segundo [Ros22], obtém-se que a Temperatura aproximado para esse elemento linear pode ser dada por,

$$T_i^{k+1} = T_i^k + (\frac{\Delta t}{\Delta x^2}) \cdot (T_{i+1}^k - 2_i^k + T_{i-1}^k), 0 \leq x \leq 2, t > 0 \tag{2}$$

Assim, aplicando esse algoritmo em Python, obtém-se um algoritmo de ordem de grandeza O(N^2). Para garantir que o método não exploda, aplicou-se a condição de Von-Neuman tal que ΔT foi tomado como sendo $\Delta T = \frac{(\Delta x)^2}{2}$. Foram traçadas, então, as curvas relativas à aproximação da função desejada para diferentes valores de tempo anteriores ao tempo final, as quais foram tomadas quando uma variável auxiliar k variava a cada certo número de valores múltiplos de 10, de modo a se verificar a convergência da Equação da Condução do Calor Unidimensional para sua solução exata. As aproximações foram tomadas como representadas por pontos de diferentes cores e a solução exata foi dada por uma linha preta.

Para o exercício 1, as condições de contorno utilizadas foram:

$$T(0,t) = T(2,t) = 0, t > 0$$

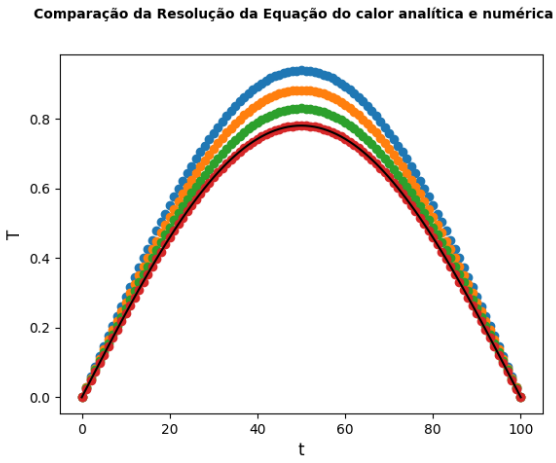
$$T(x,0) = sin(\frac{\pi}{2}x), 0 \leq x \leq 2$$

cuja solução exata é:

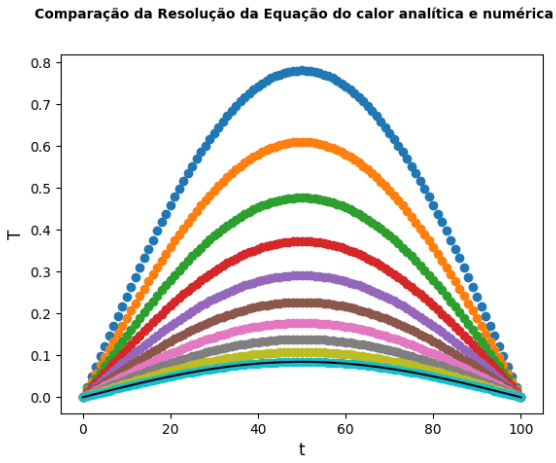
$$T(x,t) = exp[\frac{-\pi^2 t}{4}t] \cdot sin[\frac{\pi}{2}x]$$

A partir dos gráficos obtidos, 1 percebe-se que a temperatura no Elemento Linear tende a uma reta de valor zero e coeficiente angular zero quanto $t \to \infty$ e que, para valores de t pequenos, a solução tende a uma curva que possui máximo no centro do elemento, o que representa a direção do fluxo de calor quando o sistema ainda está em regime transiente tais que as condições de contorno do sistema sejam respeitadas. O sistema apresentar temperatura

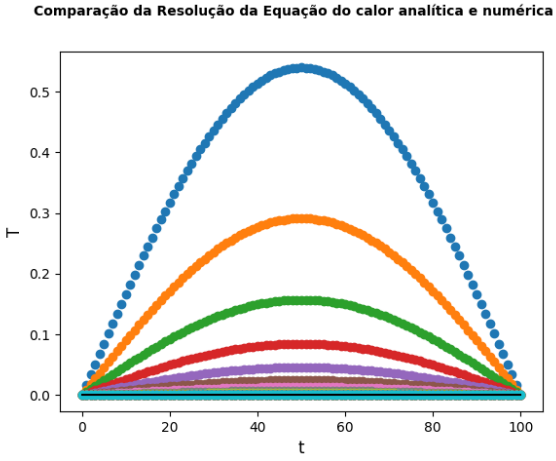
máxima em seu centro e a sua temperatura em regime permanente tender a 0 faz sentido físico ao se analisar que a temperatura entre os dois extremos da placa é 0.



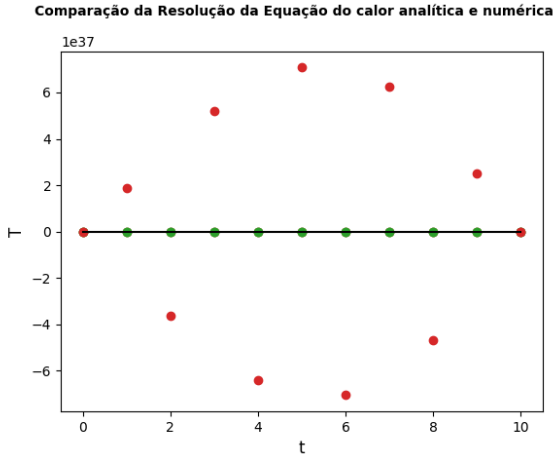
Temperatura para t = 0.1, N = 100



Temperatura para t = 1.0, N = 100



Temperatura para t = 10.0, N = 100



Caso em que, para dt = 0.25, o gráfico explode

Figura 1: Gráficos obtidos para as Curvas de Temperatura

3 Exercício 2

Tomando o Método das Diferenças Finitas para aproximar a Equação da Condução do calor para o caso unidimensional 16,

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} \tag{3}$$

Obtém-se que a Temperatura aproximado para esse elemento linear pode ser dada por,

$$T_i^{k+1} = T_i^k + (\frac{\Delta t}{\Delta x^2}) \cdot (T_{i+1}^k - 2_i^k + T_{i-1}^k), 0 \leq x \leq 1, t > 0 \tag{4}$$

Assim, de modo análogo ao realizado em 2, aplicando esse algoritmo em Python, obtém-se um algoritmo de ordem de grandeza O(N^2). Para garantir que o método não exploda, aplicou-se a condição de Von-Neuman tal que ΔT foi tomado como sendo $\Delta T = \frac{(\Delta x)^2}{2}$. Foram traçadas, então, as curvas relativas à aproximação da função desejada para diferentes valores de tempo anteriores ao tempo final, as quais foram tomadas quando uma variável auxiliar k variava a cada certo número de valores múltiplos de 10, de modo a se verificar a convergência da Equação da Condução do Calor Unidimensional para sua solução exata. As aproximações foram tomadas como representadas por pontos de diferentes cores e a solução exata foi dada por uma linha preta.

Para o exercício 2, as condições de contorno utilizadas foram:

$$T(0,t) = T(1,t) = 0, t \geq 0$$

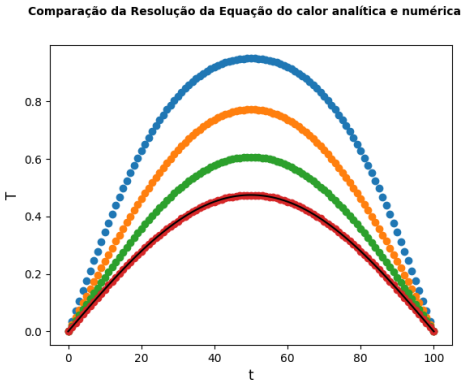
$$T(x,0) = 1, 0 \leq x \leq 1$$

cuja solução exata é:

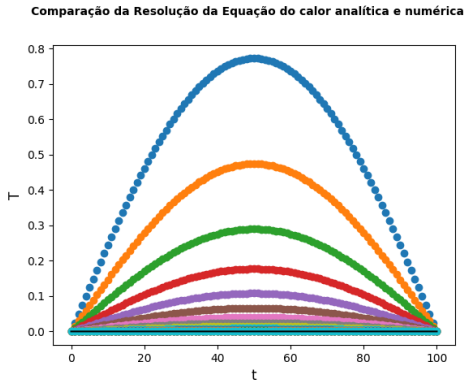
$$T(x,t) = \sum_{n=1}^{\infty}((\frac{4}{(2n-1)\pi})sin[(2n-1)\pi x]exp[-(2n-1)^2\pi^2 t])$$

É importante observar que a solução exata da equação foi realizada para 1000 valores de n, sendo, logo, uma aproximação da resposta final esperada. A partir dos gráficos obtidos, 2, percebe-se que a temperatura no Elemento Linear tende a uma reta de valor zero e coeficiente angular zero quanto $t \to \infty$ e que, para valores de t pequenos, a solução tende a uma curva que possui

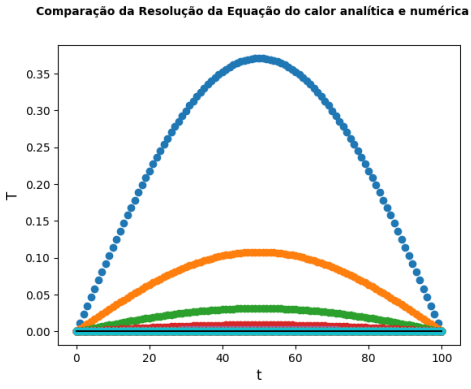
máximo no centro do elemento, o que representa a direção do fluxo de calor quando o sistema ainda está em regime transiente tais que as condições de contorno do sistema sejam respeitadas. O sistema apresentar temperatura máxima em seu centro e a sua temperatura em regime permanente tender a 0 faz sentido físico ao se analisar que a temperatura entre os dois extremos da placa é 0, sendo, logo, uma interpretação física similar à realizada para o 2. É interessante ressaltar, entretanto que o 3 possui convergência mais rápida que o 2, o que, muito provavelmente, se deve ao fato da função da Temperatura em função da variável x para o exercício 3 representar uma melhor condutividade térmica do material que a do exercício 2, tendo em vista que é constante e igual a 1, enquanto que, para 2, é um função senoidal de valor máximo 1.



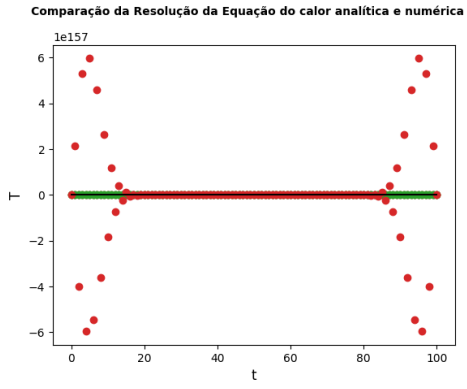
Temperatura para t = 0.1, N = 100



Temperatura para t = 1.0, N = 100



Temperatura para t = 10.0, N = 100



Caso em que, para dt = 0.25, o gráfico explode

Figura 2: Gráficos obtidos para as Curvas de Temperatura

4 Exercício 3

Tomando o Método das Diferenças Finitas para aproximar a Equação da Condução do calor para o caso unidimensional 16,

$$\frac{\partial T}{\partial t} = 0.01 \cdot \frac{\partial^2 T}{\partial x^2} \tag{5}$$

Obtém-se que a Temperatura aproximado para esse elemento linear pode ser dada por,

$$T_i^{k+1} = T_i^k + (\frac{\Delta t}{\Delta x^2}) \cdot (T_{i+1}^k - 2_i^k + T_{i-1}^k), 0 \leq x \leq 1, t > 0 \tag{6}$$

Assim, de modo análogo ao realizado em 2, aplicando esse algoritmo em Python, obtém-se um algoritmo de ordem de grandeza O(N^2). Para garantir que o método não exploda, aplicou-se a condição de Von-Neuman tal que ΔT foi tomado como sendo $\Delta T = \frac{(\Delta x)^2}{2}$. Foram traçadas, então, as curvas relativas à aproximação da função desejada para diferentes valores de tempo anteriores ao tempo final, as quais foram tomadas quando uma variável auxiliar k variava a cada certo número de valores múltiplos de 10, de modo a se verificar a convergência da Equação da Condução do Calor Unidimensional para sua solução exata. As aproximações foram tomadas como representadas por pontos de diferentes cores e a solução exata foi dada por uma linha preta.

Para o exercício 3, as condições de contorno utilizadas foram:

$$T(0,t) = 1, t > 0$$

$$T(1,t) = 0, t > 0$$

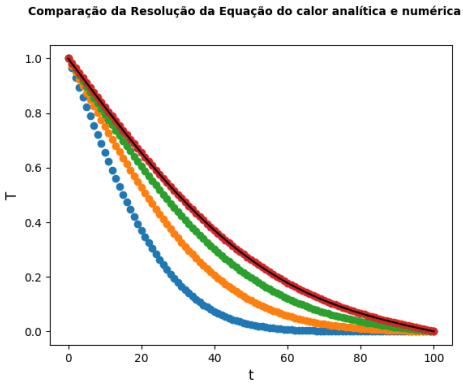
$$T(x,0) = 0, 0 \leq x \leq 1$$

cuja solução exata é:

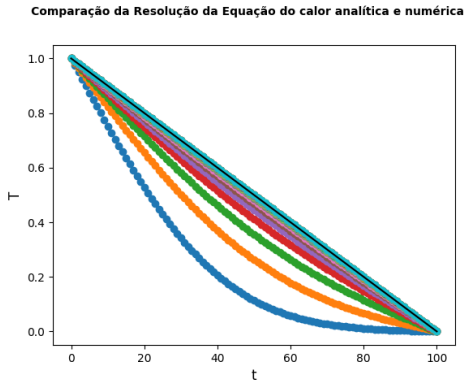
$$T(x,t) = 1 - x - \sum_{n=1}^{\infty}((\frac{2}{n\pi})sin[n\pi x]exp[-n^2\pi^2 t])$$

É importante observar que a solução exata da equação foi realizada para 1000 valores de n, sendo, logo, uma aproximação da resposta final esperada. A partir dos gráficos obtidos, 3, percebe-se que a temperatura no Elemento

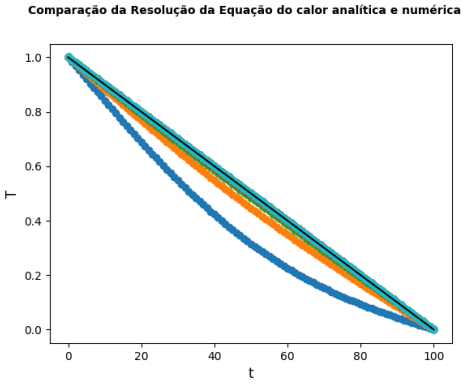
Linear tende a uma reta de coeficiente angular negativo cujo valor final é zero quanto $t \to \infty$ e que, para valores de t pequenos, a solução tende a uma curva que possui máximo no início do elemento, o que representa a direção do fluxo de calor de seu extremo mais quente (t = 0) para seu extremo mais frio (t = 1) quando o sistema ainda está em regime transiente tais que as condições de contorno do sistema sejam respeitadas. O sistema apresentar temperatura máxima em seu início e a sua temperatura em regime permanente tender a uma reta decrescente faz sentido físico ao se analisar que a diferença de temperatura entre os dois extremos do elemento é 1, logo, a temperatura dos elementos finitos tomados aumenta até o ponto em que a transferência de calor entre esses elementos seja 0, o que representa a equação de uma reta. O estado transiente ser uma curva em crescimento também faz sentido ao se analisar que, a cada iteração, a temperatura nos elementos finitos tem de se aproximar à seu valor correspondente na reta. É interessante ressaltar, entretanto que o 4 possui similar ao 3, o que, muito provavelmente, se deve ao fato da função da Temperatura em função da variável x para o exercício 3 ser igual à do exercício 4, possuindo, logo, uma forma de solução exata similar, ou seja, convergência próxima.



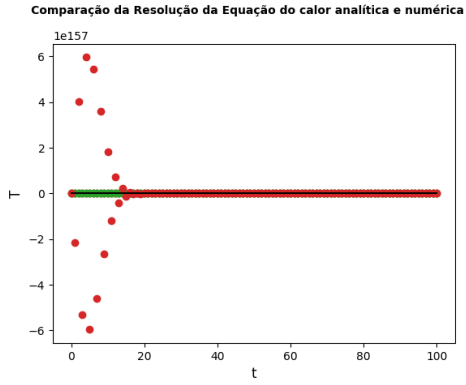
Temperatura para t = 0.1, N = 100



Temperatura para t = 1.0, N = 100



Temperatura para t = 10.0, N = 100



Caso em que, para dt = 0.25, o gráfico explode

Figura 3: Gráficos obtidos para as Curvas de Temperatura

5 Exercício 4

Tomando o Método das Diferenças Finitas para aproximar a Equação da Condução do calor para o caso unidimensional 16,

$$\frac{\partial T}{\partial t} = \frac{}{\partial^2 T}\partial x^2 \tag{7}$$

Obtém-se que a Temperatura aproximado para esse elemento linear pode ser dada por,

$$T_i^{k+1} = T_i^k + (\frac{\Delta t}{\Delta x^2}) \cdot (T_{i+1}^k - 2_i^k + T_{i-1}^k), 0 \leq x \leq 1, t > 0 \tag{8}$$

Assim, de modo análogo ao realizado em 2, aplicando esse algoritmo em Python, obtém-se um algoritmo de ordem de grandeza O(N^2). Para garantir que o método não exploda, aplicou-se a condição de Von-Neuman tal que ΔT foi tomado como sendo $\Delta T = \frac{(\Delta x)^2}{2}$. Foram traçadas, então, as curvas relativas à aproximação da função desejada para diferentes valores de tempo anteriores ao tempo final, as quais foram tomadas quando uma variável auxiliar k variava a cada certo número de valores múltiplos de 10, de modo a se verificar a convergência da Equação da Condução do Calor Unidimensional para sua solução exata. As aproximações foram tomadas como representadas por pontos de diferentes cores e a solução exata foi dada por uma linha preta.

Para o exercício 4, as condições de contorno utilizadas foram:

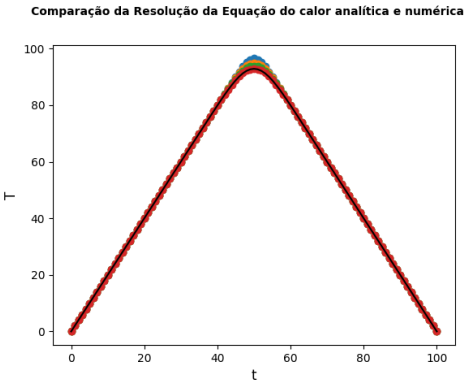
$$T(0,t) = T(1,t) = 0, t \geq 0$$

$$\begin{cases} T(x,0) = 200x, 0 \leq x \leq 0.5 \\ T(x,0) = 200(1-x), 0.5 < x \leq 1 \end{cases}$$

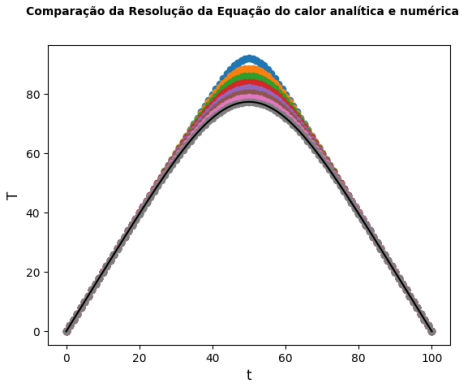
cuja solução exata é:

$$T(x,t) = (\frac{800}{\pi^2}) \cdot (\sum_{n=0}^{\infty}((\frac{(-1)^n}{(2n+1)^2})sin[(2n+1)\pi x]exp[-(2n+1)^2\pi^2 0.01t]))$$

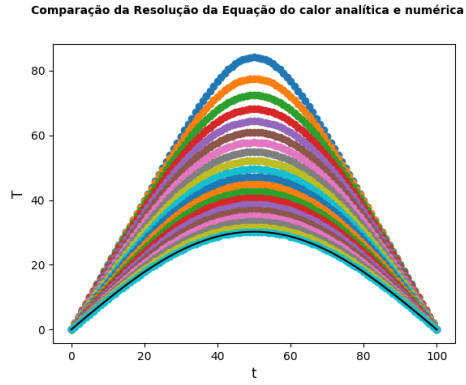
É importante observar que a solução exata da equação foi realizada para 1000 valores de n, sendo, logo, uma aproximação da resposta final esperada. A partir dos gráficos obtidos, 8, percebe-se que a temperatura no Elemento Linear tende a uma reta de valor zero e coeficiente angular zero quanto $t \to \infty$ e que, para valores de t pequenos, a solução tende a uma curva que possui máximo no centro do elemento, o que representa a direção do fluxo de calor quando o sistema ainda está em regime transiente tais que as condições de contorno do sistema sejam respeitadas, de modo análogo ao obtido em 2 e 3. O sistema apresentar temperatura máxima em seu centro e a sua temperatura em regime permanente tender a 0 faz sentido físico ao se analisar que a temperatura entre os dois extremos da placa é 0, sendo, logo, uma interpretação física similar à realizada para o 2. É interessante ressaltar, entretanto que o **??** possui convergência mais lenta que para os exercícios 2 e 3, o que se deve ao fato de que, na Equação da Condução do Calor, há um coeficiente α tal que o valor da influência da derivada parcial segunda da Temperatura em função de x seja 100 vezes menor que para os exercícios 2 e 3, o que explica sua convergência mais lenta.



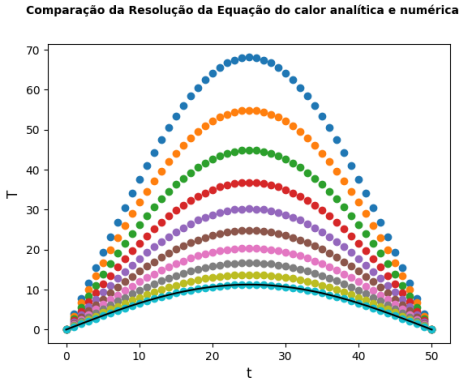
Temperatura para t = 0.1, N = 100



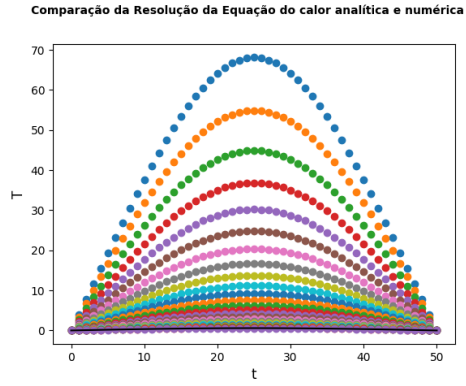
Temperatura para t = 1.0, N = 100



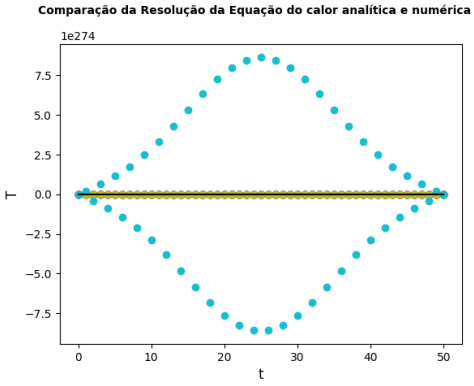
Temperatura para t = 10.0, N = 50



Temparatura para t = 20.0, N = 100



Temperatura para t = 50.0, N = 50



Caso em que, para dt = 0.25, o gráfico explode

Figura 4: Gráficos obtidos para as Curvas de Temperatura

6 Exercício 5

Tomando o Método das Diferenças Finitas para aproximar a Equação da Condução do calor para o caso unidimensional 16,

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2} \tag{9}$$

Obtém-se que a Temperatura aproximado para esse elemento linear pode ser dada por,

$$T_i^{k+1} = T_i^k + (\frac{\Delta t}{\Delta x^2}) \cdot (T_{i+1}^k - 2_i^k + T_{i-1}^k), 0 \leq x \leq 0.5, t > 0 \tag{10}$$

Assim, de modo análogo ao realizado em 2, aplicando esse algoritmo em Python, obtém-se um algoritmo de ordem de grandeza O(N^2). Para garantir que o método não exploda, aplicou-se a condição de Von-Neuman tal que ΔT foi tomado como sendo $\Delta T = \frac{(\Delta x)^2}{2}$. Foram traçadas, então, as curvas relativas à aproximação da função desejada para diferentes valores de tempo anteriores ao tempo final, as quais foram tomadas quando uma variável auxiliar k variava a cada certo número de valores múltiplos de 10, de modo a se verificar a convergência da Equação da Condução do Calor Unidimensional para sua solução exata. As aproximações foram tomadas como representadas por pontos de diferentes cores e a solução exata foi dada por uma linha preta.

Para o exercício 5, as condições de contorno utilizadas foram:

$$T(0,t), t > 0$$

$$\frac{\partial T}{\partial t}(O.5,t) = 0$$

$$T(x,0) = 200x, 0 \leq x \leq 0.5$$

cuja solução exata é:

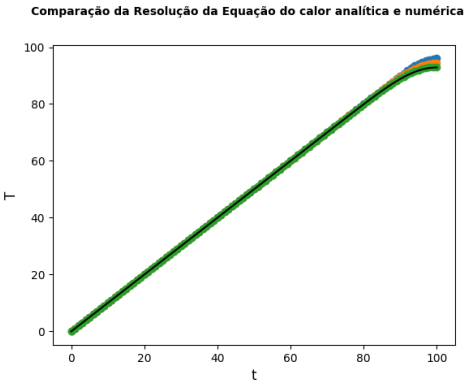
$$T(x,t) = (\frac{800}{\pi^2}) \cdot (\sum_{n=0}^{\infty}((\frac{(-1)^n}{(2n+1)^2})sin[(2n+1)\pi x]exp[-(2n+1)^2\pi^2 0.01t]))$$

É importante observar que a solução exata da equação foi realizada para 1000 valores de n, sendo, logo, uma aproximação da resposta final esperada. A partir dos gráficos obtidos, 5, percebe-se que a temperatura no Elemento

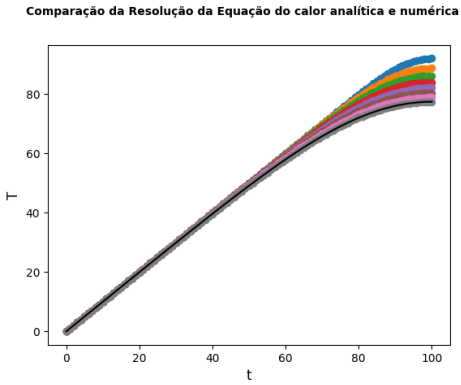
Linear tende a uma curva de valor mínimo 0 e valor máximo 1 quanto $t \to \infty$ e que, para valores de t pequenos, a solução tende a uma reta de coeficiente angular positivo, valor mínimo 0 e valor máximo 1, o que representa a direção do fluxo de calor quando o sistema ainda está em regime transiente tais que as condições de contorno do sistema sejam respeitadas, de modo análogo ao obtido em 2 e 3. O sistema apresentar temperatura máxima em seu último valor e a sua temperatura em regime permanente tender a 1 faz sentido físico ao se analisar que a temperatura final representa o elemento central da curva obtida no exercício 5, sendo, logo, o valor máximo da curva.

Para se implementar a condição de contorno de Neuman, no ponto x =0.5, utilizou-se o método do Ghost-Point, segundo consta em [Ros22], ou seja, aplicou-se o algoritmo:

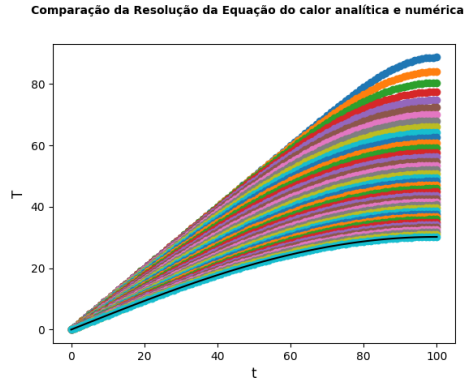
$$T_5^{k+1} = T_5^k + (\frac{\Delta t}{\Delta x^2}) \cdot (T_4^k - 2T_5^k) \tag{11}$$



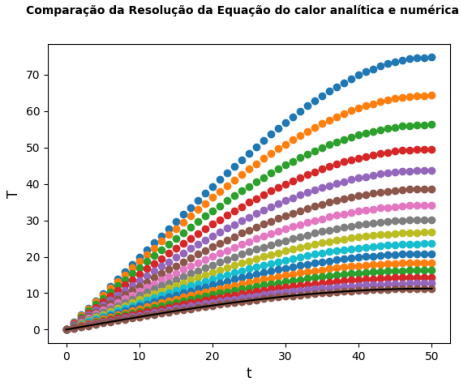
Temperatura para t = 0.1, N = 100



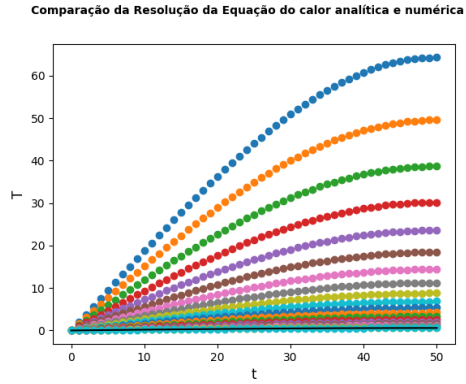
Temperatura para t = 1.0, N = 100



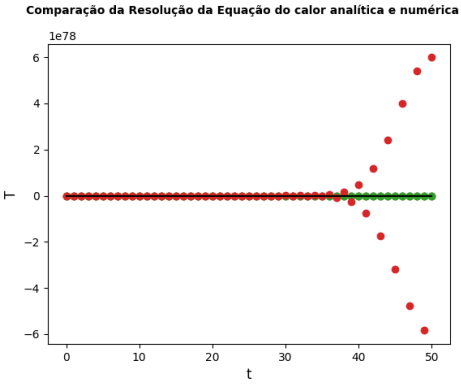
Temperatura para t = 10.0, N = 100



Temparatura para t = 20.0, N = 50



Temperatura para t = 50.0, N = 50



Caso em que, para dt = 0.25, o gráfico explode

Figura 5: Gráficos obtidos para as Curvas de Temperatura

7 Exercício 6

Tomando o Método das Diferenças Finitas para aproximar a Equação da Condução do calor para o caso bidimensional 16,

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2}\frac{\partial^2 T}{\partial y^2} \tag{12}$$

Obtém-se que a Temperatura aproximado para uma placa pode ser dada por,

$$T_{i,j}^{k+1} = T_{i,j}^{k} + (\frac{\Delta t}{\Delta x^2}) \cdot (T_{i+1,j}^{k} - 2_{i,j}^{k} + T_{i-1,j}^{k}) + (\frac{\Delta t}{\Delta y^2}) \cdot (T_{i,j+1}^{k} - 2_{i,j}^{k} + T_{i,j-1}^{k}), 0 \leq x \leq 10 \leq y \leq 1, t \geq 0 \tag{13}$$

Logo, aplicando esse algoritmo em Python, obtém-se um algoritmo de ordem de grandeza O(N^2). Não foram aplicados meios de garantir que o método não exploda segundo o valor de dt, ainda que Δt tenha sido determinado como $\frac{\Delta x}{4}$. Foram traçadas, então, curvas relativas à solução exata da função (linhas pretas) para 5 valores de x e y diferentes, cujo intervalo desejado foi dividido em 5 partes iguais para os valores máximos e mínimos de x e de y, correspondendo aos valores de temperatura em pontos da placa para o tempo final desejado. Para a solução aproximada, foram traçadas linhas coloridas para os mesmos valores de x, y e tempo utilizados para a solução exata. Variou-se, então, o tempo final e o número de passos para verificar o comportamento da equação aproximada com sua solução exata para o tempo final tomado. É importante ressaltar que, para o algoritmo numérico tomado, a temperatura nas quinas da placa não importam para o cálculo das aproximações desejadas, já que pontos na diagonal não são considerados.

Dessa forma, para o exercício 6, as condições de contorno utilizadas foram:

$$T(x, 0, t) = 0, 0 \leq x \leq 1 t \geq 0$$

$$T(x, 1, t) = sin(\pi x), 0 \leq x \leq 1, t \geq 0$$

$$T(0, y, t) = 0, 0 \leq y \leq 1, t \geq 0$$

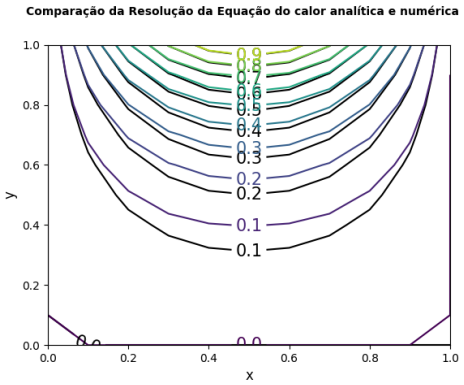
$$T(1, y, t) = 0, 0 < y < 1, t \geq 0$$

$$T(x, y, 0) = 0, 0 < x < 1, 0 < y < 1$$

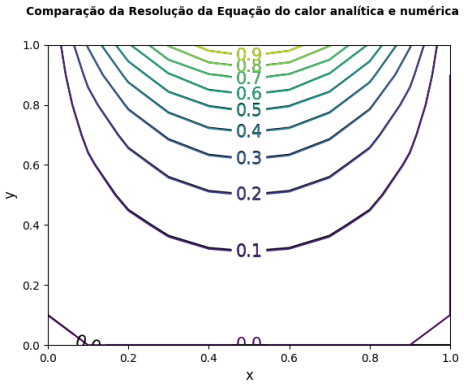
cuja solução exata é:

$$T(x, t \to \infty) = \frac{sinh(\pi y) sin(\pi x)}{sinh(\pi)}$$

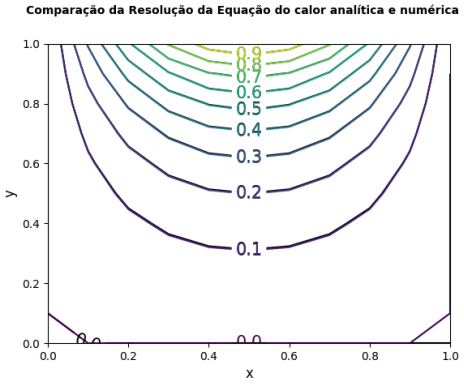
A partir dos gráficos obtidos,6, percebe-se que a temperatura na placa tende a curvas de valores mínimos nas bordas laterais da placa e máximos em seu centro para os intervalos requisitados em x e y quando $t \to \infty$, tal que, conforme se aumentam os valores de x e y, a temperatura tende a seu máximo devido às condições de contorno tomadas. É importante ressaltar que, para x = y = 0, a temperatura na placa é 0, porém, devido à precisão númerica de 16 casas decimais do Python, esse valor é representado com um pequeno erro no gráfico. Para valores de t pequenos, as temperaturas obtidas são representadas com maior imprecisão que em relação à sua solução analítica, representando, logo, que, conforme $t \to \infty$, a solução numérica tende à solução exata. A temperatura do sistema em regime permanente tender a seu máximo quando y tender a 1 faz sentido físico ao se analisar que essa é a temperatura determinada, logo, máxima, na borda superior da placa. É interessante ressaltar que a temperatura parte de seu zero, chega a seu máximo, e volta a seu zero do ponto de vista da borda lateral da placa e, do ponto de vista da sua borda superior, sua temperatura aumenta até seus valores máximos determinados por suas condições de contorno.



Temperatura para t = 0.1, N = 10



Temperatura para t = 1.0, N = 10



Temperatura para t = 10.0, N = 10

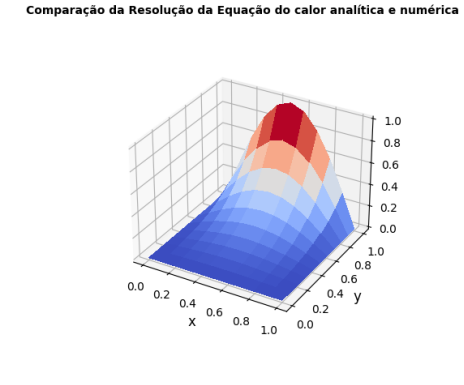


Gráfico 3D de Temperatura para t = 0.1, N = 10

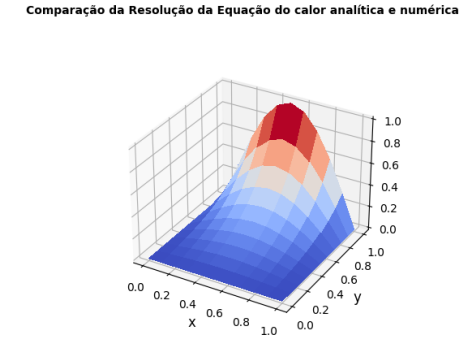


Gráfico 3D de Temperatura para t = 1.0, N = 10

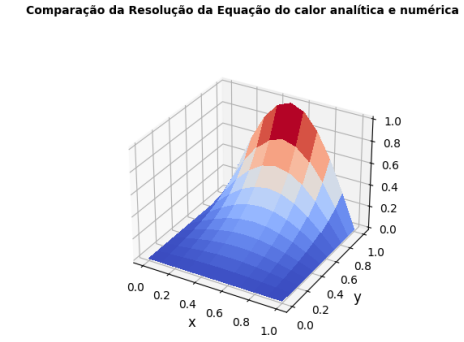


Gráfico 3D de Temperatura para t = 10.0, N = 10

Figura 6: Gráficos obtidos para as Curvas de Temperatura

8 Exercício 7

Tomando o Método das Diferenças Finitas para aproximar a Equação da Condução do calor para o caso bidimensional 16,

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2}\frac{\partial^2 T}{\partial y^2} \tag{14}$$

Obtém-se que a Temperatura aproximado para uma placa pode ser dada por,

$$T_{i,j}^{k+1} = T_{i,j}^{k} + (\frac{\Delta t}{\Delta x^2})\cdot(T_{i+1,j}^{k} - 2_{i,j}^{k} + T_{i-1,j}^{k}) + (\frac{\Delta t}{\Delta y^2})\cdot(T_{i,j+1}^{k} - 2_{i,j}^{k} + T_{i,j-1}^{k}), 0 \leq x \leq 0.50 \leq y \leq 1, t \geq 0 \tag{15}$$

Logo, aplicando esse algoritmo em Python, obtém-se um algoritmo de ordem de grandeza O(N^2). Não foram aplicados meios de garantir que o método não exploda segundo o valor de dt, ainda que Δt tenha sido determinado como $\frac{\Delta x}{4}$. Foram traçadas, então, curvas relativas à solução exata da função (linhas pretas) para 5 valores de x e y diferentes, cujo intervalo desejado foi dividido em 5 partes iguais para os valores máximos e mínimos de x e de y, correspondendo aos valores de temperatura em pontos da placa para o tempo final desejado. Para a solução aproximada, foram traçadas linhas coloridas para os mesmos valores de x, y e tempo utilizados para a solução exata. Variou-se, então, o tempo final e o número de passos para verificar o comportamento da equação aproximada com sua solução exata para o tempo final tomado. É importante ressaltar que, para o algoritmo numérico tomado, a temperatura nas quinas da placa não importam para o cálculo das aproximações desejadas, já que pontos na diagonal não são considerados.

Dessa forma, para o exercício 7, as condições de contorno utilizadas foram:

$$T(x, 0, t) = 0, 0 \leq x \leq 0.5t \geq 0$$

$$T(x, 1, t) = sin(\pi x), 0 \leq x \leq 1, t \geq 0$$

$$T(0, y, t) = 0, 0 \leq y \leq 1, t \geq 0$$

$$T(1, y, t) = 0, 0 < y < 1, t \geq 0$$

$$T(x, y, 0) = 0, 0 < x < 1, 0 < y < 1$$

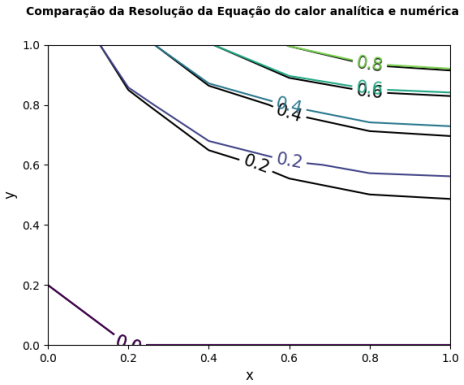
cuja solução exata é:

$$T(x, t \to \infty) = \frac{sinh(\pi y)sin(\pi x)}{sinh(\pi)}$$

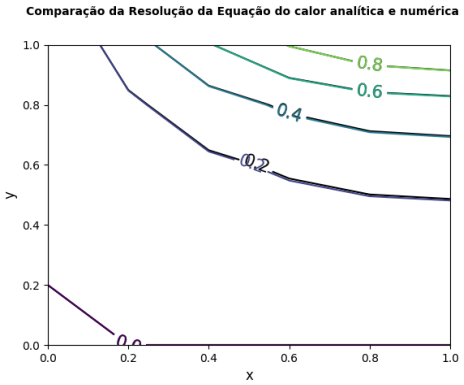
A partir dos gráficos obtidos, 7, percebe-se que a temperatura na placa tende a uma curva crescente para os intervalos requisitados em x e y quando $t \to \infty$, tal que, conforme se aumentam os valores de x e y, a temperatura tende a seu máximo devido às condições de contorno tomadas. É importante ressaltar que, para x = y = 0, a temperatura na placa é 0, porém, devido à precisão númerica de 16 casas decimais do Python, esse valor é representado com um pequeno erro no gráfico. Para valores de t pequenos, as temperaturas obtidas são representadas com maior imprecisão que em relação à sua solução analítica, representando, logo, que, conforme $t \to \infty$, a solução numérica tende à solução exata. A temperatura do sistema em regime permanente tender a 1 quando y tender a $sin(\pi x)$ faz sentido físico ao se analisar que essa é a temperatura na borda superior da placa. Esse problema representa a metade do problema realizado em 7, logo, para o valor de x máximo (x = 0.5), a temperatura deve tender a seu valo máximo, conforme pode ser visto no gráfico, que representa, também, metade do gráfico realizado em 7. Nesse caso, a temperatura na borda esquerda do gráfico não é mais zero devido às suas condições de contorno, assim como sua temperatura em sua borda superior, de modo que a temperatura aumenta conforme x e y aumentam. Para se realizar a condiçao de contorno de Neumann de T em relação a x que representa a condição de contorno em x =0.5, aplicou-se o método da aproximação de Segunda Ordem, de modo a se realizar as equações:

$$T_N = \frac{4}{3}T_4 - \frac{1}{3}T_3$$

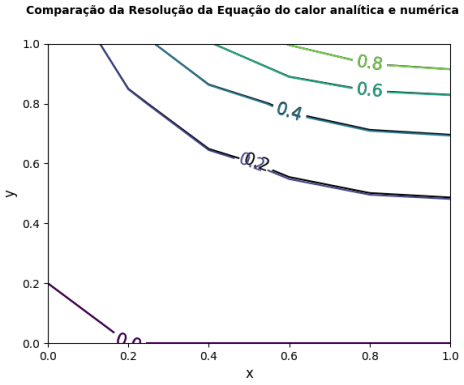
Para a aproximação numérica, considerou-se apenas a variação da Temperatura em x, não em y.



Temperatura para t = 0.1, N = 10



Temperatura para t = 1.0, N = 10



Temperatura para t = 10.0, N = 10

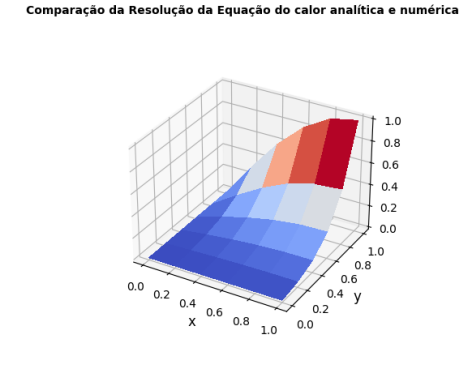


Gráfico 3D de Temperatura para t = 0.1, N = 10

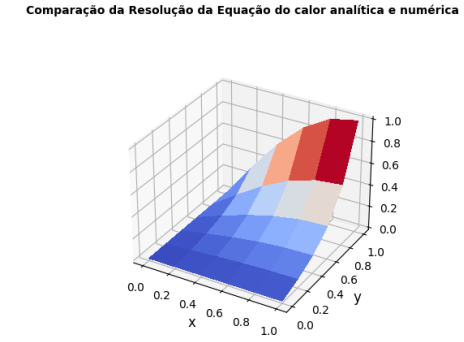


Gráfico 3D de Temperatura para t = 1.0, N = 10

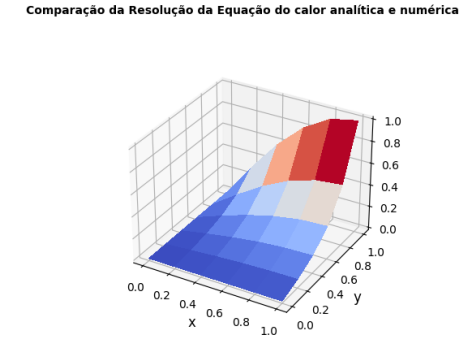


Gráfico 3D de Temperatura para t = 10.0, N = 10

Figura 7: Gráficos obtidos para as Curvas de Temperatura

9 Exercício 8

Tomando o Método das Diferenças Finitas para aproximar a Equação da Condução do calor para o caso bidimensional 16,

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2}\frac{\partial^2 T}{\partial y^2} \tag{16}$$

Obtém-se que a Temperatura aproximado para uma placa pode ser dada por,

$$T_{i,j}^{k+1} = T_{i,j}^{k} + (\frac{\Delta t}{\Delta x^2})\cdot(T_{i+1,j}^{k} - 2_{i,j}^{k} + T_{i-1,j}^{k}) + (\frac{\Delta t}{\Delta y^2})\cdot(T_{i,j+1}^{k} - 2_{i,j}^{k} + T_{i,j-1}^{k}), 0 \leq x \leq 10 \leq y \leq 1, t > 0 \tag{17}$$

É importante observar que a solução exata da equação foi realizada para 10 valores de n, sendo, logo, uma aproximação da resposta final esperada. Logo, aplicando esse algoritmo em Python, obtém-se um algoritmo de ordem de grandeza O(N^2). Não foram aplicados meios de garantir que o método não exploda segundo o valor de dt, ainda que Δt tenha sido determinado como $\frac{\Delta x}{4}$. Foram traçadas, então, curvas relativas à solução exata da função (linhas pretas) para 5 valores de x e y diferentes, cujo intervalo desejado foi dividido em 5 partes iguais para os valores máximos e mínimos de x e de y, correspondendo aos valores de temperatura em pontos da placa para o tempo final desejado. Para a solução aproximada, foram traçadas linhas coloridas para os mesmos valores de x, y e tempo utilizados para a solução exata. Variou-se, então, o tempo final e o número de passos para verificar o comportamento da equação aproximada com sua solução exata para o tempo final tomado. É importante ressaltar que, para o algoritmo numérico tomado, a temperatura nas quinas da placa não importam para o cálculo das aproximações desejadas, já que pontos na diagonal não são considerados.

Dessa forma, para o exercício 8, as condições de contorno utilizadas foram:

$$T(x, O, t) = 0, 0 \leq x \leq 1 t \geq 0$$

$$\begin{cases} T(x, 1, t) = 75x, 0 \leq x \leq \frac{2}{3}, t \geq 0 \\ T(x, 1, t) = 150(1 - x), \frac{2}{3} < x \leq 1 \geq 0 \end{cases}$$

$$T(0, y, t) = 0, 0 \leq y \leq 1, t \geq 0$$

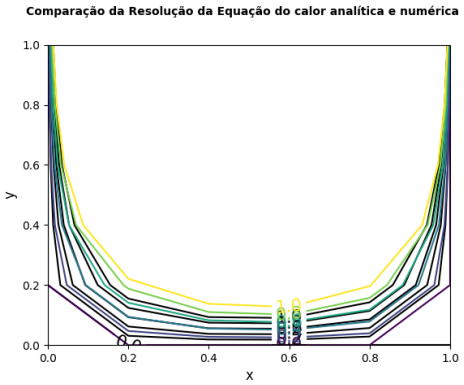
$$T(1, y, t) = 0, 0 < y < 1, t \geq 0$$

$$T(x, y, 0) = 0, 0 < x < 1, 0 < y < 1$$

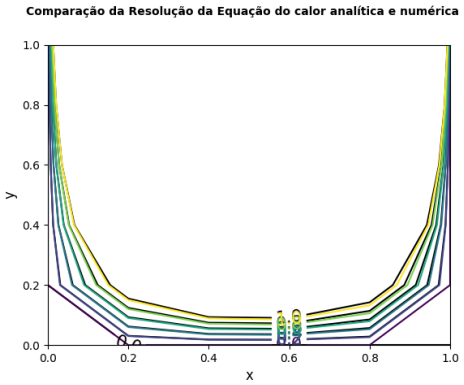
cuja solução exata é:

$$T(x,t) = (\frac{450}{\pi^2}) \cdot (\sum_{n=1}^{\infty}((\frac{sin(\frac{2n\pi}{3})}{n^2 sinh(n\pi)})sin(n\pi x)sinh(n\pi x)))$$

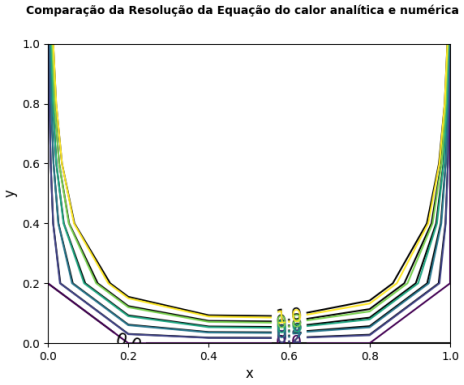
A partir dos gráficos obtidos, 8, percebe-se que a temperatura na placa tende a uma curva crescente que depois decresce até 0 para os intervalos requisitados em x e y quando $t \to \infty$, tal que, conforme se aumentam os valores de x e y, a temperatura tende a seu máximo devido às condições de contorno tomadas. É importante ressaltar que, para x = y = 0, a temperatura na placa é 0, porém, devido à precisão númerica de 16 casas decimais do Python, esse valor é representado com um pequeno erro no gráfico. Para valores de t pequenos, as temperaturas obtidas são representadas com maior imprecisão que em relação à sua solução analítica, representando, logo, que, conforme $t \to \infty$, a solução numérica tende à solução exata. A temperatura do sistema em regime permanente tender a 0 nas bordas laterais faz sentido físico, já que a temperatura nessas bordas é 0. Os valores de x e y tomados apresentam uma curva de crescimento mais aguda para suas bordas laterais que as obtidas em 7, o que se deve às condições de contorno tomadas em sua borda superior apresentarem uma variação muito mais aguda que a tomada para o exercício 7.



Temperatura para t = 0.1, N = 10



Temperatura para t = 1.0, N = 10



Temperatura para t = 10.0, N = 10

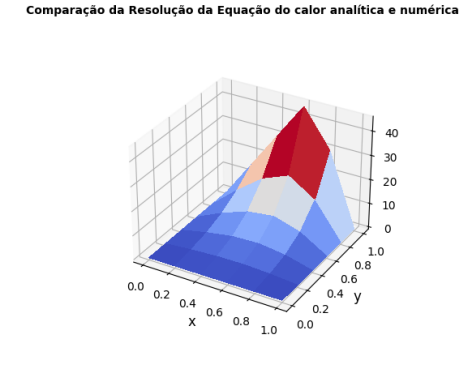


Gráfico 3D de Temperatura para t = 0.1, N = 10

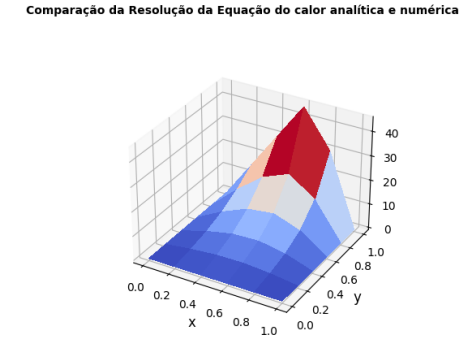


Gráfico 3D de Temperatura para t = 1.0, N = 10

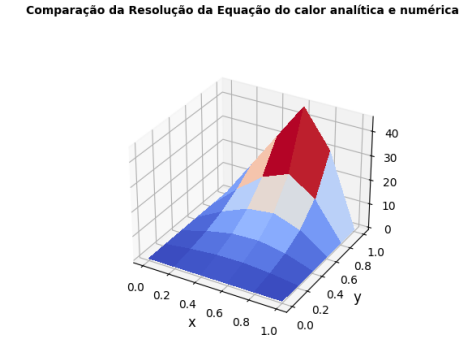
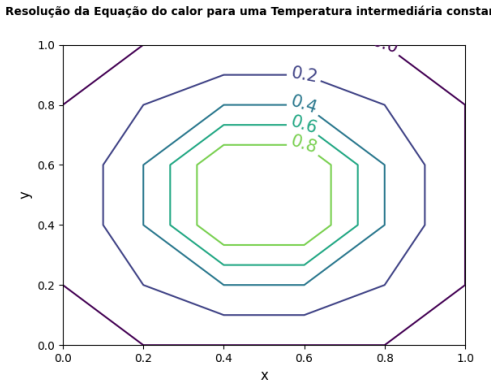


Gráfico 3D de Temperatura para t = 10.0, N = 10

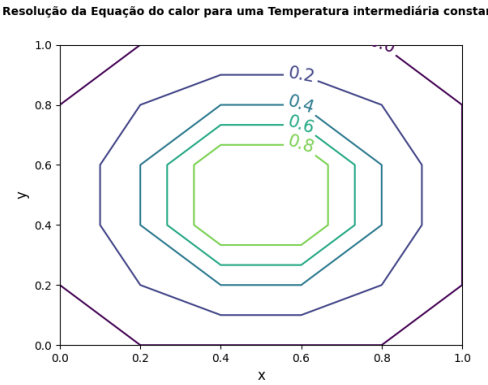
Figura 8: Gráficos obtidos para as Curvas de Temperatura

10 Exercício 9

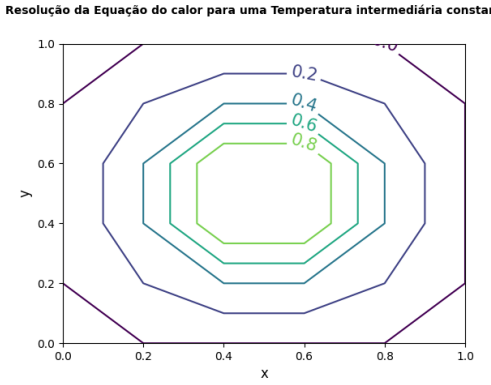
Conforme pode ser observado em 9, para se resolver a Equação do Calor Bidimensional para uma placa com um "obstáculo"de temperatura constante, utilizou-se uma amostragem de 5 passos de modo a se considerar os valores de 0,4 a 0,6 como sendo constantes e iguais às condições de contorno do elemento de temperatura constante tomada. Assim, adicionando essa condicional e procedendo a resolução da equação de condução de calor bidimensional conforme realizado em 7, percebeu-se que o gráfico da equação da condução bidimensional comporta-se como os gráficos obtidos em 7, porém, contornando o obstáculo. Logo, o fluxo de calor obtido das curvas isotérmicas traçadas contorna o obstáculo como um fluido escoando em um tubo contorna um obstáculo colocado em seu centro, de modo que a temperatura seja maior próximas ao obstáculos e diminuam até 0, conforme pode ser visto nos gráficos obtidos.



Temperatura para t = 0.1, N = 5



Temperatura para t = 1.0, N = 5



Temperatura para t = 10.0, N = 5

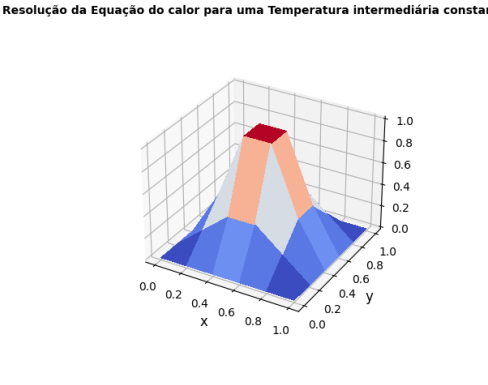


Gráfico 3D de Temperatura para t = 0.1, N = 5

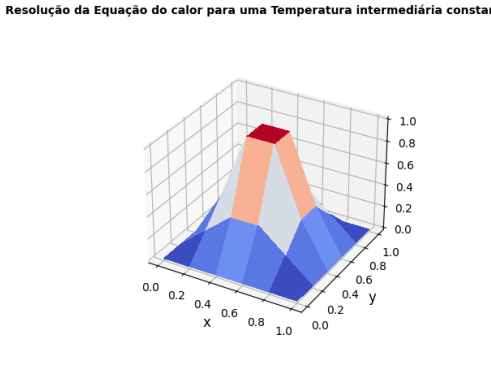


Gráfico 3D de Temperatura para t = 1.0, N = 5

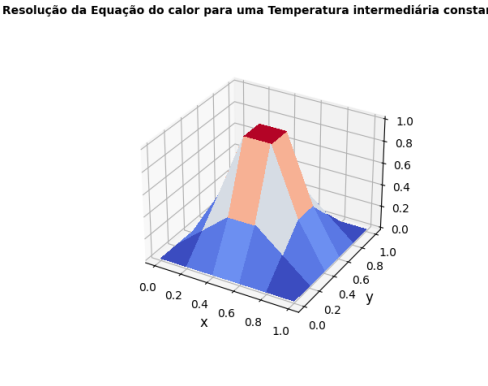


Gráfico 3D de Temperatura para t = 10.0, N = 5

Figura 9: Gráficos obtidos para as Curvas de Temperatura

11 Exercício 10

Para se resolver o sistema linear,

$$\begin{bmatrix} 12 & -2 & 3 & 1 \\ 1 & 6 & 20 & -4 \\ -2 & 15 & 6 & -3 \\ 0 & -3 & 2 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 20 \\ 0 \\ 0 \end{bmatrix}$$

Foi necessário inverter sua segunda e terceira linha de modo que o sistema seja diagonalmente dominante e, logo, para que seja possível aplicar os Métodos Numéricos de Jacobi, Gauss-Seidel e SOR. Resolveu-se, assim, a seguinte matriz para uma precisa de 10^-8,

$$\begin{bmatrix} 12 & -2 & 3 & 1 \\ -2 & 15 & 6 & -3 \\ 1 & 6 & 20 & -4 \\ 0 & -3 & 2 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 20 \\ 0 \\ 0 \end{bmatrix}$$

Resolvendo-se o Sistema, obteve-se a seguinte solução pelo método de Jacobi:

$$\begin{bmatrix} -0.33439845 \\ -0.57219291 \\ 1.10128558 \\ -0.4354611 \end{bmatrix}$$

Pelo método de Gauss-Seidel:

$$\begin{bmatrix} -0.33439846 \\ -0.57219291 \\ 1.10128558 \\ -0.4354611 \end{bmatrix}$$

Pelo método SOR com coeficiente igual a 1:

$$\begin{bmatrix} -0.33439845 \\ -0.57219291 \\ 1.10128558 \\ -0.4354611 \end{bmatrix}$$

12 Exercício 11

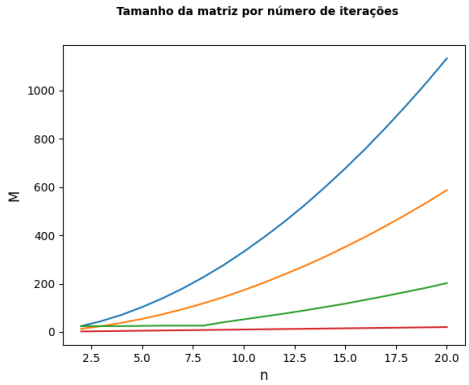
Para se realizar os gráficos, foi tomado um n máximo de 20 devido ao custo computacional para se realizar para n maiores. Mediu-se, então, a partir de uma variável de acumulação, o número máximo de iterações; a partir do módulo time do python, o tempo transcorrido nas iterações e, a

partir de uma array do numpy r, pode-se obter os valores de R máximo. Esses valores foram, então, reorganizados em três gráficos diferentes de modo a se comparar os métodos numéricos para diferentes critérios. O primeiro critério tomado foi o número de iterações por n, seguido do tempo transcorrido por n, ambos de n = 2 até n = 20 tomado em passo de 1 e, finalmente, de R máximo em relação ao número de iterações realizado para n = 20. Ao se tomar os gráficos do tamanho da matriz n por seu número de iterações (M), 10, percebe-se que o método de Jacobi requer mais iterações de acordo com o tamanho da matriz e que apresentam crescimento maior que para os métodos de Gauss-Seidel, SOR e Gradiente Conjugado. Para o método de Gauss Seidel e SOR (ω tomado como 1.5), a convergência é similar ao método de Jacobi, porém, atenuada, de modo a apresentar convergência mais rápida que o método de Jacobi. Para o método do Gradiente Conjugado, o número de iterações cresce linearmente com o tamanho da matriz, conforme pode ser observado no segundo gráfico, sendo, em comparação com os outros métodos, praticamente constante. O método do Gradiente Conjugado requer, logo, um menor número de iterações para convergir que os outros métodos numéricos utilizados, sendo seguido pelos métodos de SOR (dependendo do valor tomado para ω), Gauss-Seidel e o Método de Jacobi, o qual requer o maior número de iterações para convergir.

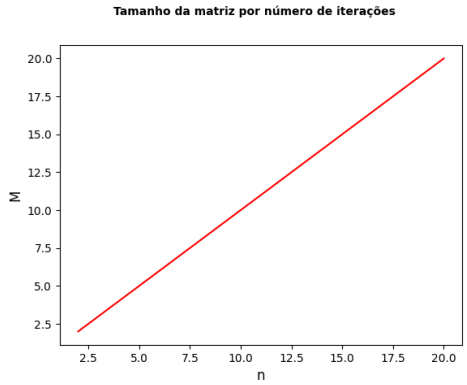
Em relação ao tempo necessário para a convergência dos gráficos segundo o tamanho da Matriz, percebe-se que o método de Jacobi cresce de modo similar a uma exponencial, apresentando, logo, um custo temporal maior que os métodos de Gauss-Seidel, SOR e Gradiente Conjugado. Para os métodos de Gauss-Seidel e SOR, a convergência em relação ao tempo cresce de modo similar ao Jacobi, porém, de modo atenuado, sendo menor para o método SOR devido ao valor de $\omega = 1.5$ tomado. O Método do Gradiente Conjugado apresenta convergência quase constante, porém, ao se observar apenas esse método, percebe-se um custo de tempo constante, seguido de uma variação rápida e uma nova estabilização, a qual pode ser tanto característica do método de gradiente conjugado ou uma distribuição aleatória de alguns casos em que o Método converge mais lentamente.

Em relação ao número máximo de R em relação ao número de iterações, percebe0se que, para os métodos de Jacobi, Gauss-Seidel e SOR, o gráfico é extremamente similar, de modo que R começa tendendo ao infinito, ou seja, grande e tende a um valor de 0, apresentando uma distribuição parecida com a de uma função hiperbólica. Para o método do Gradiente Conjugado, R começa muito pequeno e aumenta de modo aleatório, com períodos de

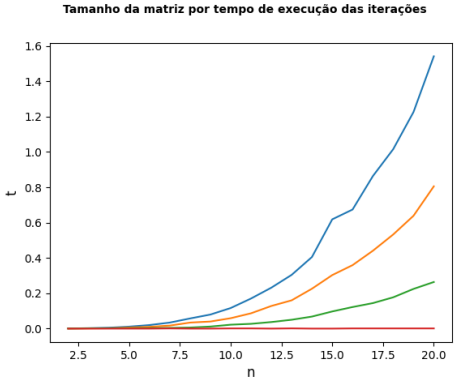
crescimento e decrescimento, até a convergência do método.



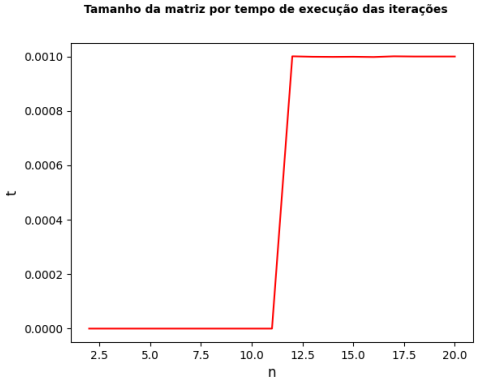
Iterações por n



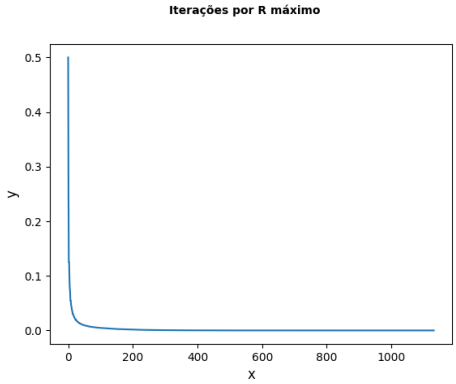
Iterações por n para o gradiente conjugado



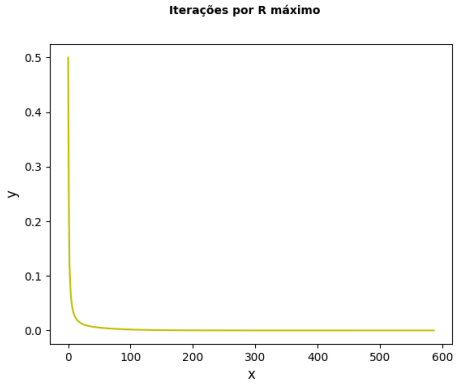
Tempo por n



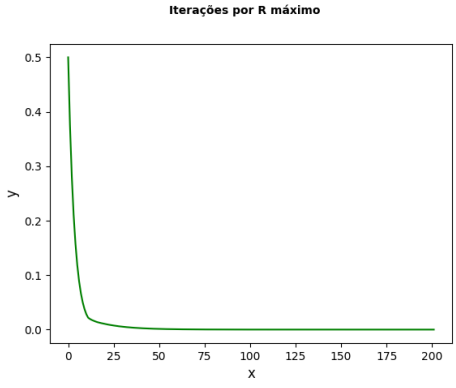
Tempo por n para o gradiente conjugado



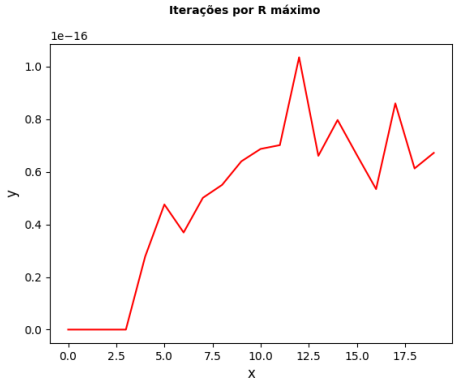
R máximo por iterações para Jacobi



R máximo por iterações para Gauss-Seidel



R máximo por iterações para SOR



R máximo por iterações para Gradiente Conjugado

A Código Exercício 1

```
import numpy as np
import matplotlib.pyplot as plt

def f(t,x):
    return np.exp(((-np.pi**2)*t)/4)*np.sin((np.pi/2)*x)

def T_i(T,x,dx):
    T[0] = 0.0
    T[N] = 0.0
    for i in range(1,N):
        x+=dx
        T[i] = np.sin((np.pi/2.0)*x)
    return T

def solexata(N,T,tempo_final):
    x_aux = 0.0
    for i in range(1,N):
        x_aux+=dx
        T[i] = f(tempo_final,x_aux)

    plt.plot(T,'k')

def eqcondcal(N,dx,dt,tempo_final,T):
    tempo = 0.0

    c = dt/(dx*dx)
    k = 0

    while tempo <= tempo_final:
        T_new = np.copy(T)
        for i in range(1,N):

            T_new[i] = T[i] + c*(T[i+1]-2.0*T[i]+T[i-1])

        tempo+=dt
        k+=1
        T = np.copy(T_new)
```

```
        if k%10 == 0:
            plt.plot(T,'o')



N = 10 #numero de passos
dx = 2.0/N
dt = (dx*dx)/4
tempo_final = 1.0

k = 0

if dt > (dx*dx)/2: #condicao de estabilidade de Von Neuman
    print('Erro')

else:
    T = np.zeros(N+1)
    T_exata = np.zeros(N+1)
    x = 0.0
    T = T_i(T,x,dx)

    T_exata[0] = 0.0
    T_exata[N] = 0.0

    #Inicializando o Grafico

    graf = plt.figure()
    ax = graf.add_subplot()

    # Parmetros do Grfico

    graf.suptitle(' Comparao da Resoluo da Equao do calor
        analtica e numrica ', fontsize = 10, fontweight = 'bold',
        usetex = False)
    ax.set_ylabel(r'T', fontsize = 12, usetex = False)
    ax.set_xlabel(r't', fontsize = 12, usetex = False)

    #Calculando a Equacao da Conducao do Calor

    eqcondcal(N,dx,dt,tempo_final,T)
```

```
    solexata(N,T_exata,tempo_final)

    plt.show()
```

B Código Exercício 2

```
import numpy as np
import matplotlib.pyplot as plt

def f(t,x):
    res = 0.0
    for n in range(1,1000+1):
        res+=
            (4/((2*n-1)*np.pi))*np.sin((2*n-1)*np.pi*x)*np.exp(-((2*n-1)**2)*(np.pi**2)*t)
    return res

def T_i(T,x,dx):
    T[0] = 0.0
    T[N] = 0.0
    for i in range(1,N):
        x+=dx
        T[i] = 1.0
    return T

def solexata(N,T,tempo_final):
    x_aux = 0.0
    for i in range(1,N):
        x_aux+=dx
        T[i] = f(tempo_final,x_aux)

    plt.plot(T,'k')

def eqcondcal(N,dx,dt,tempo_final,T):

    tempo = 0.0

    c = dt/(dx*dx)
    k = 0
```

```
    while tempo <= tempo_final:
        T_new = np.copy(T)
        for i in range(1,N):

            T_new[i] = T[i] + c*(T[i+1]-2.0*T[i]+T[i-1])

        tempo+=dt
        k+=1
        T = np.copy(T_new)
        if k%10 == 0:
            plt.plot(T,'o')



N = 10 #numero de passos
dx = 1.0/N
dt = (dx*dx)/2
tempo_final = 10.0
k = 0

if dt > (dx*dx)/2: #condicao de estabilidade de Von Neuman
    print('Erro')

else:
    T = np.zeros(N+1)
    T_exata = np.zeros(N+1)
    x = 0.0
    T = T_i(T,x,dx)

    T_exata[0] = 0.0
    T_exata[N] = 0.0

    #Inicializando o Grafico

    graf = plt.figure()
    ax = graf.add_subplot()

    #Parmetros do Grafico
```

```
    graf.suptitle(' Comparao  da  Resoluo  da  Equao  do calor
        analtica e  numrica ', fontsize = 10, fontweight = 'bold',
        usetex = False)
    ax.set_ylabel(r'T', fontsize = 12, usetex = False)
    ax.set_xlabel(r't', fontsize = 12, usetex = False)


    #Calculando a Equacao da Conducao do Calor

    eqcondcal(N,dx,dt,tempo_final,T)
    solexata(N,T_exata,tempo_final)

    plt.show()
```

C Código Exercício 3

```
import numpy as np
import matplotlib.pyplot as plt

def f(t,x):
    res = 0.0
    for n in range(1,1000+1):
        res+=((2/(n*np.pi))*(np.sin(n*np.pi*x)*np.exp(-(n**2)*(np.pi**2)*t)))
    return 1- x - res

def T_i(T,x,dx):
    T[0] = 1.0
    T[N] = 0.0
    for i in range(1,N):
        x+=dx
        T[i] = 0.0
    return T

def solexata(N,T,tempo_final):
    x_aux = 0.0
    for i in range(1,N):
        x_aux+=dx
        T[i] = f(tempo_final,x_aux)
```

```
    plt.plot(T,'k')

def eqcondcal(N,dx,dt,tempo_final,T):
    tempo = 0.0

    c = dt/(dx*dx)
    k = 0

    while tempo <= tempo_final:
        T_new = np.copy(T)
        for i in range(1,N):

            T_new[i] = T[i] + c*(T[i+1]-2.0*T[i]+T[i-1])

        tempo+=dt
        k+=1
        T = np.copy(T_new)
        if k%10 == 0:
            plt.plot(T,'o')



N = 10 #numero de passos
dx = 1.0/N
dt = (dx*dx)/2
tempo_final = 1.0
k = 0

if dt > (dx*dx)/2: #condicao de estabilidade de Von Neuman
    print('Erro')

else:
    T = np.zeros(N+1)
    T_exata = np.zeros(N+1)
    x = 0.0
    T = T_i(T,x,dx)

    T_exata[0] = 1.0
    T_exata[N] = 0.0
```

```
    #Inicializando o Grafico

    graf = plt.figure()
    ax = graf.add_subplot()

    # Parmetros  do Grafico

    graf.suptitle(' Comparao  da  Resoluo  da  Equao  do calor
        analtica e  numrica ', fontsize = 10, fontweight = 'bold',
        usetex = False)
    ax.set_ylabel(r'T', fontsize = 12, usetex = False)
    ax.set_xlabel(r't', fontsize = 12, usetex = False)

    #Calculando a Equacao da Conducao do Calor

    eqcondcal(N,dx,dt,tempo_final,T)
    solexata(N,T_exata,tempo_final)

    plt.show()
```

D Código Exercício 4

```
import numpy as np
import matplotlib.pyplot as plt

def f(t,x):
    res = 0.0
    for n in range(1000+1):
        res+=
            (((-1)**n)/((2*n+1)**2))*np.sin((2*n+1)*np.pi*x)*np.exp(-((2*n+1)**2)*(np.pi**
    return (800/(np.pi**2))*res

def T_i(T,x,dx):
    T[0] = 0.0
    T[N] = 0.0
    for i in range(1,N):
        x+=dx
```

```
        if x <= 0.5:
            T[i] = 200*x
        else:
            T[i] = 200*(1-x)
    return T

def solexata(N,T,tempo_final):
    x_aux = 0.0
    for i in range(1,N):
        x_aux+=dx
        T[i] = f(tempo_final,x_aux)

    plt.plot(T,'k')

def eqcondcal(N,dx,dt,tempo_final,T):
    tempo = 0.0

    c = dt/(dx*dx)
    k = 0

    while tempo <= tempo_final:
        T_new = np.copy(T)
        for i in range(1,N):

            T_new[i] = T[i] + 0.01*c*(T[i+1]-2.0*T[i]+T[i-1])

        tempo+=dt
        k+=1
        T = np.copy(T_new)
        if k%10 == 0:
            plt.plot(T,'o')



N = 10 #numero de passos
dx = 1.0/N
dt = (dx*dx)/2
tempo_final = 20.0
k = 0
```

```
if dt > (dx*dx)/2: #condicao de estabilidade de Von Neuman
    print('Erro')

else:
    T = np.zeros(N+1)
    T_exata = np.zeros(N+1)
    x = 0.0
    T = T_i(T,x,dx)

    T_exata[0] = 0.0
    T_exata[N] = 0.0

    #Inicializando o Grafico

    graf = plt.figure()
    ax = graf.add_subplot()

    # Parmetros do Grafico

    graf.suptitle(' Comparao da Resoluo da Equao do calor
        analtica e numrica ', fontsize = 10, fontweight = 'bold',
        usetex = False)
    ax.set_ylabel(r'T', fontsize = 12, usetex = False)
    ax.set_xlabel(r't', fontsize = 12, usetex = False)

    #Calculando a Equacao da Conducao do Calor

    eqcondcal(N,dx,dt,tempo_final,T)
    solexata(N,T_exata,tempo_final)

    plt.show()
```

E Código Exercício 5

```
import numpy as np
import matplotlib.pyplot as plt

def f(t,x):
```

```
    res = 0.0
    for n in range(100):
        res+=
            (((-1)**n)/((2*n+1)**2))*np.sin((2*n+1)*np.pi*x)*np.exp(-((2*n+1)**2)*(np.pi**
    return (800/(np.pi**2))*res

def T_i(T,x,dx):
    T[0] = 0.0
    for i in range(1,N+1):
        x+=dx
        T[i] = 200*x
    return T

def solexata(N,T,tempo_final):
    x_aux = 0.0
    for i in range(1,N+1):
        x_aux+=dx
        T[i] = f(tempo_final,x_aux)

    plt.plot(T,'k')

def eqcondcal(N,dx,dt,tempo_final,T):
    tempo = 0.0

    c = dt/(dx*dx)
    k = 0

    while tempo <= tempo_final:
        T_new = np.copy(T)
        for i in range(1,N+1):
            #ghost-point
            if i == N:
                T_new[i] = T[i] + 0.01*c*(2.0*T[i-1]-2.0*T[i])

            else:
                T_new[i] = T[i] + 0.01*c*(T[i+1]-2.0*T[i]+T[i-1])


        tempo+=dt
        k+=1
```

```
        T = np.copy(T_new)
        if k%100 == 0:
            plt.plot(T,'o')



N = 10 #numero de passos
dx = 0.5/N
dt = (dx*dx)/2
tempo_final = 1.0
k = 0

if dt > (dx*dx)/2: #condicao de estabilidade de Von Neuman
    print('Erro')

else:
    T = np.zeros(N+1)
    T_exata = np.zeros(N+1)
    x = 0.0
    T = T_i(T,x,dx)

    T_exata[0] = 0.0

    #Inicializando o Grafico

    graf = plt.figure()
    ax = graf.add_subplot()

    #Parmetros do Grafico

    graf.suptitle('Comparao da Resoluo da Equao do calor
        analtica e numrica', fontsize = 10, fontweight = 'bold',
        usetex = False)
    ax.set_ylabel(r'T', fontsize = 12, usetex = False)
    ax.set_xlabel(r't', fontsize = 12, usetex = False)

    #Calculando a Equacao da Conducao do Calor

    eqcondcal(N,dx,dt,tempo_final,T)
    solexata(N,T_exata,tempo_final)
```

```
    plt.show()
```

F Código Exercício 6

```
import numpy as np
import matplotlib.pyplot as plt

def f(t,x,y):
    return (np.sinh(np.pi*y)*np.sin(np.pi*x))/(np.sinh(np.pi))

def cond_iniciais(Nx,Ny,dx,dy,T):
    #Esquerda
    y_aux = 0.0
    for j in range(Ny+1):
        T[0,j] = 0.0
        y_aux+=dy
    #Direita
    y_aux = 0.0
    for j in range(Ny+1):
        T[Nx,j] = 0.0
        y_aux+=dy
    #Baixo
    x_aux = 0.0
    for i in range(Nx+1):
        T[i,0] = 0.0
        x_aux+=dx
    #Cima
    x_aux = 0.0
    for i in range(Nx+1):
        T[i,Ny] = np.sin(np.pi*x_aux)
        x_aux+=dx
    #quinas nao sao utilizadas para os calculos
    return T

def solexata(Nx,Ny,T,tempo_final):
    x_aux = 0.0
    y_aux = 0.0
```

```
    intervalos = np.linspace(0.0,1.0,Nx+1)
    T_x = np.linspace(0.0,1.0,Nx+1)
    T_y = np.linspace(0.0,1.0,Ny+1)
    for i in range(Nx+1):
        y_aux = 0.0
        for j in range(Ny+1):
            T[i,j] = f(tempo_final,x_aux,y_aux)
            y_aux+=dy
        x_aux+=dx
    c = plt.contour(T_x,T_y,np.transpose(T),intervalos, colors = 'k')
    plt.clabel(c, inline = True, fontsize = 15, colors = 'k')

def eqcondcal(Nx,Ny,dx,dt,tempo_final,T):
    tempo = 0.0
    intervalos = np.linspace(0.0,1.0,Nx+1)
    T_x = np.linspace(0.0,1.0,Nx+1)
    T_y = np.linspace(0.0,1.0,Ny+1)
    T_new = np.copy(T)
    while tempo <= tempo_final:

        for i in range(1,Nx):
            for j in range(1,Ny):
                T_new[i,j] = T[i,j] +
                    (dt/(dx*dx))*(T[i+1,j]-2.0*T[i,j]+T[i-1,j]) +
                    (dt/(dy*dy))*(T[i,j+1]-2.0*T[i,j]+T[i,j-1])
        T = np.copy(T_new)
        tempo+=dt

    c = plt.contour(T_x,T_y,np.transpose(T),intervalos)
    plt.clabel(c, inline = True, fontsize = 15)

Nx = 5
Ny = 5
dx = 1.0/Nx
dy = 1.0/Ny
dt = (dx*dx)/4.0
tempo_final = 1.0

T = np.zeros((Nx+1,Nx+1),float)
T_exata = np.zeros((Nx+1,Nx+1),float)
```

```
T = cond_iniciais(Nx,Ny,dx,dy,T)

tempo = 0.0
tempo_final = 1.0

#Inicializando o Grafico

graf = plt.figure()
ax = graf.add_subplot()

# Parmetros do Grafico

graf.suptitle(' Comparao da Resoluo da Equao do calor analtica
    e numrica', fontsize = 10, fontweight = 'bold', usetex = False)
ax.set_ylabel(r'y', fontsize = 12, usetex = False)
ax.set_xlabel(r'x', fontsize = 12, usetex = False)

#Calculando a Equacao da Conducao do Calor

solexata(Nx,Ny,T_exata,tempo_final)
eqcondcal(Nx,Ny,dx,dt,tempo_final,T)
plt.show() #Inicializando o Grafico
```

G Código Exercício 7

```
import numpy as np
import matplotlib.pyplot as plt

def f(t,x,y):
    return (np.sinh(np.pi*y)*np.sin(np.pi*x))/(np.sinh(np.pi))

def cond_iniciais(Nx,Ny,dx,dy,T):
    #Esquerda
    y_aux = 0.0
    for j in range(Ny+1):
        T[0,j] = 0.0
        y_aux+=dy
    #Direita por aproximacao de segunda ordem
```

```
    y_aux = 0.0
    for j in range(Ny+1):
        T[Nx,j] = (4/3)*T[Nx-1,j] -(1/3)*T[Nx-2,j]
        y_aux+=dy
    #Baixo
    x_aux = 0.0
    for i in range(Nx+1):
        T[i,0] = 0.0
        x_aux+=dx
    #Cima
    x_aux = 0.0
    for i in range(Nx+1):
        T[i,Ny] = np.sin(np.pi*x_aux)
        x_aux+=dx
    #quinas nao sao utilizadas para os calculos
    return T

def solexata(Nx,Ny,T,tempo_final):
    x_aux = 0.0
    y_aux = 0.0
    intervalos = np.linspace(0.0,1.0,Nx+1)
    T_x = np.linspace(0.0,1.0,Nx+1)
    T_y = np.linspace(0.0,1.0,Ny+1)
    for i in range(Nx+1):
        y_aux = 0.0
        for j in range(Ny+1):
            T[i,j] = f(tempo_final,x_aux,y_aux)
            y_aux+=dy
        x_aux+=dx
    c = plt.contour(T_x,T_y,np.transpose(T),intervalos, colors = 'k')
    plt.clabel(c, inline = True, fontsize = 15, colors = 'k')

def eqcondcal(Nx,Ny,dx,dt,tempo_final,T):
    tempo = 0.0
    intervalos = np.linspace(0.0,1.0,Nx+1)
    T_x = np.linspace(0.0,1.0,Nx+1)
    T_y = np.linspace(0.0,1.0,Ny+1)
    T_new = np.copy(T)
    while tempo <= tempo_final:
```

```
        for i in range(1,Nx+1):
            for j in range(1,Ny):
                if i == Nx:
                    T_new[i,j] = (4/3)*T[Nx-1,j] -(1/3)*T[Nx-2,j]
                else:
                    T_new[i,j] = T[i,j] +
                        (dt/(dx*dx))*(T[i+1,j]-2.0*T[i,j]+T[i-1,j]) +
                        (dt/(dy*dy))*(T[i,j+1]-2.0*T[i,j]+T[i,j-1])

        T = np.copy(T_new)
        tempo+=dt

    c = plt.contour(T_x,T_y,np.transpose(T),intervalos)
    plt.clabel(c, inline = True, fontsize = 15)

Nx = 5
Ny = 5
dx = 0.5/Nx
dy = 1.0/Ny
dt = (dx*dx)/4.0

T = np.zeros((Nx+1,Nx+1),float)
T_exata = np.zeros((Nx+1,Nx+1),float)
T = cond_iniciais(Nx,Ny,dx,dy,T)

tempo = 0.0
tempo_final = 0.1

#Inicializando o Grafico

graf = plt.figure()
ax = graf.add_subplot()

# Parmetros do Grafico

graf.suptitle('Comparao da Resoluo da Equao do calor analtica
    e numrica', fontsize = 10, fontweight = 'bold', usetex = False)
ax.set_ylabel(r'y', fontsize = 12, usetex = False)
ax.set_xlabel(r'x', fontsize = 12, usetex = False)
```

```
#Calculando a Equacao da Conducao do Calor

solexata(Nx,Ny,T_exata,tempo_final)
eqcondcal(Nx,Ny,dx,dt,tempo_final,T)
plt.show() #Inicializando o Grafico
```

H Código Exercício 8

```
import numpy as np
import matplotlib.pyplot as plt

def f(t,x,y):
    res = 0.0
    for n in range(1,11):
        res+=((np.sin(2*n*np.pi/3))/(n*n*np.sinh(n*np.pi)))*np.sin(n*np.pi*x)*np.sinh(n*np
    return (450/(np.pi*np.pi))*res

def cond_iniciais(Nx,Ny,dx,dy,T):
    #Esquerda
    y_aux = 0.0
    for j in range(Ny+1):
        T[0,j] = 0.0
        y_aux+=dy
    #Direita
    y_aux = 0.0
    for j in range(Ny+1):
        T[Nx,j] = 0.0
        y_aux+=dy
    #Baixo
    x_aux = 0.0
    for i in range(Nx+1):
        T[i,0] = 0.0
        x_aux+=dx
    #Cima
    x_aux = 0.0
    for i in range(Nx+1):
        if x_aux <= (2/3):
            T[i,Ny] = 75*x_aux
```

```
        else:
            T[i,Ny] = 150*(1-x_aux)
        x_aux+=dx
    #quinas nao sao utilizadas para os calculos
    return T

def solexata(Nx,Ny,T,tempo_final):
    x_aux = 0.0
    y_aux = 0.0
    intervalos = np.linspace(0.0,1.0,Nx+1)
    T_x = np.linspace(0.0,1.0,Nx+1)
    T_y = np.linspace(0.0,1.0,Ny+1)
    for i in range(Nx+1):
        y_aux = 0.0
        for j in range(Ny+1):
            T[i,j] = f(tempo_final,x_aux,y_aux)
            y_aux+=dy
        x_aux+=dx
    c = plt.contour(T_x,T_y,np.transpose(T),intervalos, colors = 'k')
    plt.clabel(c, inline = True, fontsize = 15, colors = 'k')

def eqcondcal(Nx,Ny,dx,dt,tempo_final,T):
    tempo = 0.0
    intervalos = np.linspace(0.0,1.0,Nx+1)
    T_x = np.linspace(0.0,1.0,Nx+1)
    T_y = np.linspace(0.0,1.0,Ny+1)
    T_new = np.copy(T)
    while tempo <= tempo_final:

        for i in range(1,Nx):
            for j in range(1,Ny):
                T_new[i,j] = T[i,j] +
                    (dt/(dx*dx))*(T[i+1,j]-2.0*T[i,j]+T[i-1,j]) +
                    (dt/(dy*dy))*(T[i,j+1]-2.0*T[i,j]+T[i,j-1])
        T = np.copy(T_new)
        tempo+=dt

    c = plt.contour(T_x,T_y,np.transpose(T),intervalos)
    plt.clabel(c, inline = True, fontsize = 15)
```

```
Nx = 5
Ny = 5
dx = 1.0/Nx
dy = 1.0/Ny
dt = (dx*dx)/4.0
tempo_final = 0.1

T = np.zeros((Nx+1,Nx+1),float)
T_exata = np.zeros((Nx+1,Nx+1),float)
T = cond_iniciais(Nx,Ny,dx,dy,T)

tempo = 0.0
tempo_final = 1.0

#Inicializando o Grafico

graf = plt.figure()
ax = graf.add_subplot()

# Parmetros do Grafico

graf.suptitle(' Comparao da Resoluo da Equao do calor analtica
    e numrica ', fontsize = 10, fontweight = 'bold', usetex = False)
ax.set_ylabel(r'y', fontsize = 12, usetex = False)
ax.set_xlabel(r'x', fontsize = 12, usetex = False)

#Calculando a Equacao da Conducao do Calor

solexata(Nx,Ny,T_exata,tempo_final)
eqcondcal(Nx,Ny,dx,dt,tempo_final,T)
plt.show() #Inicializando o Grafico
```

I Código Exercício 9

```
import numpy as np
import matplotlib.pyplot as plt

def cond_iniciais(Nx,Ny,dx,dy,T):
```

```
    x_aux = 0.0
    y_aux = 0.0
    for i in range(Nx+1):
        y_aux = 0.0
        for j in range(Ny+1):
            if (i == 2 or i == 3) and (j == 2 or j == 3):
                T[i,j] = 1.0
            else:
                T[i,j] = 0.0
            y_aux+=dy
        x_aux+=dx
    return T

def eqcondcal(Nx,Ny,dx,dt,tempo_final,T):
    tempo = 0.0
    x_aux = 0.0
    y_aux = 0.0
    intervalos = np.linspace(0.0,1.0,Nx+1)
    T_x = np.linspace(0.0,1.0,Nx+1)
    T_y = np.linspace(0.0,1.0,Ny+1)
    T_new = np.copy(T)
    while tempo <= tempo_final:
        for i in range(1,Nx):
            for j in range(1,Ny):
                if (i == 2 or i == 3) and (j == 2 or j == 3):
                    T_new[i,j] = 1.0
                else:
                    T_new[i,j] = T[i,j] +
                        (dt/(dx*dx))*(T[i+1,j]-2.0*T[i,j]+T[i-1,j]) +
                        (dt/(dy*dy))*(T[i,j+1]-2.0*T[i,j]+T[i,j-1])

        T = np.copy(T_new)
        tempo+=dt

    c = plt.contour(T_x,T_y,np.transpose(T),intervalos)
    plt.clabel(c, inline = True, fontsize = 15)

Nx = 5
Ny = 5
dx = 1.0/Nx
```

```
dy = 1.0/Ny
dt = (dx*dx)/4.0
tempo_final = 0.1

T = np.zeros((Nx+1,Nx+1),float)
T = cond_iniciais(Nx,Ny,dx,dy,T)
tempo = 0.0
tempo_final = 0.1

#Inicializando o Grafico

graf = plt.figure()
ax = graf.add_subplot()

# Parmetros do Grafico

graf.suptitle(' Resoluo da Equao do calor para uma Temperatura
    intermediria constante', fontsize = 10, fontweight = 'bold',
    usetex = False)
ax.set_ylabel(r'y', fontsize = 12, usetex = False)
ax.set_xlabel(r'x', fontsize = 12, usetex = False)

#Calculando a Equacao da Conducao do Calor
eqcondcal(Nx,Ny,dx,dt,tempo_final,T)
plt.show() #Inicializando o Grafico
```

J Código Exercício 10

```
import numpy as np
import matplotlib.pyplot as plt

#Jacobi
def jacobi(A,x,R,b,atol):
    x_novo = np.zeros(4)
    while True:
        for i in range(4):
            s = 0.0
            for j in range(4):
```

```
                s = s + A[i,j]*x[j]
            R[i] = (1.0/A[i,i])*(b[i] - s)
            x_novo[i] = x[i] + R[i]

        k+=1
        x = np.copy(x_novo)
        if np.sqrt(np.dot(R,R)) < atol:
            break

    return x
#Gauss Seidel
def gauss(A,x,R,b,atol):
    while True:
        for i in range(4):
            s = 0.0
            for j in range(4):
                s = s + A[i,j]*x[j]
            R[i] = (1.0/A[i,i])*(b[i]-s)
            x[i] = x[i] + R[i]

        if np.sqrt(np.dot(R,R)) < atol:
            break

    return x

#SOR
def sor(A,x,R,b,atol):
    w = 1 #alterar w para maxima convergencia
    while True:
        for i in range(4):
            s = 0.0
            for j in range(4):
                s = s + A[i,j]*x[j]
            R[i] = (1.0/A[i,i])*(b[i]-s)
            x[i] = x[i] + w*R[i]

        if np.sqrt(np.dot(R,R)) < atol:
            break

    return x
```

```
A =
    np.array([[12.0,-2.0,3.0,1.0],[-2.0,15.0,6.0,-3.0],[1.0,6.0,20.0,-4.0],[0.0,-3.0,2.0,9
b = np.array([0.0,0.0,20.0,0.0])
x = np.zeros(4,float)
R = np.zeros(4,float)
atol = 1e-8
print(jacobi(A,x,R,b,atol))
print(gauss(A,x,R,b,atol))
print(sor(A,x,R,b,atol))
```

K Código Exercício 11

K.1 Parte 1

```
import numpy as np
import matplotlib.pyplot as plt
import time

def mat_n(n):
    m = np.repeat([np.zeros(n,int)],n, axis = 0)
    for i in range(n):
        for j in range(n):
            if j == i-1 or j == i+1:
                m[i][j] = 1
            elif j == i:
                m[i][j] = -2
            else:
                m[i][j] = 0
    return m

#Jacobi
def jacobi(A,b,atol,n):
    x = np.zeros(n,float)
    R = np.zeros(n,float)
    x_novo = np.zeros(n)
    k = 0
    while True:
```

```
        for i in range(n):
            s = 0.0
            for j in range(n):
                s = s + A[i,j]*x[j]
            R[i] = (1.0/A[i,i])*(b[i] - s)
            x_novo[i] = x[i] + R[i]
        x = np.copy(x_novo)
        k+=1
        if np.sqrt(np.dot(R,R)) < atol:
            break

    return x,k
#Gauss Seidel
def gauss(A,b,atol,n):
    k = 0
    x = np.zeros(n,float)
    R = np.zeros(n,float)
    while True:
        for i in range(n):
            s = 0.0
            for j in range(n):
                s = s + A[i,j]*x[j]
            R[i] = (1.0/A[i,i])*(b[i]-s)
            x[i] = x[i] + R[i]

        k+=1
        if np.sqrt(np.dot(R,R)) < atol:
            break


    return x,k

#SOR
def sor(A,b,atol,n):
    k = 0
    x = np.zeros(n,float)
    R = np.zeros(n,float)
    w = 1.5 #alterar w para maxima convergencia
    while True:
        for i in range(n):
```

```
            s = 0.0
            for j in range(n):
                s = s + A[i,j]*x[j]
            R[i] = (1.0/A[i,i])*(b[i]-s)
            x[i] = x[i] + w*R[i]

        k+=1
        if np.sqrt(np.dot(R,R)) < atol:
            break


    return x,k

#metodo gradiente conjugado
def gradconj(A,b,atol,n):
    k = 0
    x = np.zeros(n,float)
    R = np.zeros(n,float)
    aux = np.dot(A,x)
    for i in range(n):
        R[i] = b[i] - aux[i]
    p = np.copy(R)
    while True:
        s = np.matmul(A,p)
        a = (np.dot(R,R))/(np.dot(p,s))
        R_aux = np.copy(R)
        for i in range(n):
            x[i] = x[i] + a*p[i]
            R[i] = R[i] - a*(s[i])

        k+=1
        if np.sqrt(np.dot(R,R)) < atol:
            break

        b = (np.dot(R,R))/(np.dot(R_aux,R_aux))
        for i in range(n):
            p[i] = R[i] + b*p[i]
    return x,k

atol = 1e-7
```

```
n = 20

k_jacobi = np.zeros(n-1,int)
k_gauss = np.zeros(n-1,int)
k_sor = np.zeros(n-1,int)
k_gradconj = np.zeros(n-1,int)

for i in range(2,n+1):
    A = mat_n(i)
    b = np.zeros(i)
    b[i-1] = -1
    x_aux, k_jacobi[i-2] = jacobi(A,b,atol,i)
    x_aux, k_gauss[i-2] = gauss(A,b,atol,i)
    x_aux, k_sor[i-2] = sor(A,b,atol,i)
    x_aux, k_gradconj[i-2] = gradconj(A,b,atol,i)
n = np.linspace(2,n,n-1)

plt.plot(n,k_jacobi)
plt.plot(n,k_gauss)
plt.plot(n,k_sor)
plt.plot(n,k_gradconj)
plt.show()
```

K.2 Parte 2

```
import numpy as np
import matplotlib.pyplot as plt
import time

def mat_n(n):
    m = np.repeat([np.zeros(n,int)],n, axis = 0)
    for i in range(n):
        for j in range(n):
            if j == i-1 or j == i+1:
                m[i][j] = 1
            elif j == i:
                m[i][j] = -2
            else:
```

```
                m[i][j] = 0
    return m

#Jacobi
def jacobi(A,b,atol,n):
    x = np.zeros(n,float)
    R = np.zeros(n,float)
    x_novo = np.zeros(n)
    t1 = time.time()
    while True:
        for i in range(n):
            s = 0.0
            for j in range(n):
                s = s + A[i,j]*x[j]
            R[i] = (1.0/A[i,i])*(b[i] - s)
            x_novo[i] = x[i] + R[i]
        x = np.copy(x_novo)
        if np.sqrt(np.dot(R,R)) < atol:
            break
    t2 = time.time()
    t = t2 - t1
    return x,t
#Gauss Seidel
def gauss(A,b,atol,n):
    x = np.zeros(n,float)
    R = np.zeros(n,float)
    t1 = time.time()
    while True:
        for i in range(n):
            s = 0.0
            for j in range(n):
                s = s + A[i,j]*x[j]
            R[i] = (1.0/A[i,i])*(b[i]-s)
            x[i] = x[i] + R[i]

        if np.sqrt(np.dot(R,R)) < atol:
            break
    t2 = time.time()
    t = t2-t1
```

```
    return x,t

#SOR
def sor(A,b,atol,n):
    x = np.zeros(n,float)
    R = np.zeros(n,float)
    w = 1.5 #alterar w para maxima convergencia
    t1 = time.time()
    while True:
        for i in range(n):
            s = 0.0
            for j in range(n):
                s = s + A[i,j]*x[j]
            R[i] = (1.0/A[i,i])*(b[i]-s)
            x[i] = x[i] + w*R[i]

        if np.sqrt(np.dot(R,R)) < atol:
            break
    t2 = time.time()
    t = t2-t1

    return x,t

#metodo gradiente conjugado
def gradconj(A,b,atol,n):
    k = 0
    x = np.zeros(n,float)
    R = np.zeros(n,float)
    aux = np.dot(A,x)
    for i in range(n):
        R[i] = b[i] - aux[i]
    p = np.copy(R)
    t1 = time.time()
    while True:
        s = np.matmul(A,p)
        a = (np.dot(R,R))/(np.dot(p,s))
        R_aux = np.copy(R)
        for i in range(n):
            x[i] = x[i] + a*p[i]
            R[i] = R[i] - a*(s[i])
```

```
        if np.sqrt(np.dot(R,R)) < atol:
            break
        b = (np.dot(R,R))/(np.dot(R_aux,R_aux))
        for i in range(n):
            p[i] = R[i] + b*p[i]
    t2 = time.time()
    t = t2-t1

    return x,t

atol = 1e-7
n = 20

t_jacobi = np.zeros(n-1,int)
t_gauss = np.zeros(n-1,int)
t_sor = np.zeros(n-1,int)
t_gradconj = np.zeros(n-1,int)

for i in range(2,n+1):
    A = mat_n(i)
    b = np.zeros(i)
    b[i-1] = -1
    x_aux, t_jacobi[i-2] = jacobi(A,b,atol,i)
    x_aux, t_gauss[i-2] = gauss(A,b,atol,i)
    x_aux, t_sor[i-2] = sor(A,b,atol,i)
    x_aux, t_gradconj[i-2] = gradconj(A,b,atol,i)
n = np.linspace(2,n,n-1)

plt.plot(n,t_jacobi)
plt.plot(n,t_gauss)
plt.plot(n,t_sor)
plt.plot(n,t_gradconj)
plt.show()
```

K.3 Parte 3

```
import numpy as np
```

```
import matplotlib.pyplot as plt

def mat_n(n):
    m = np.repeat([np.zeros(n,int)],n, axis = 0)
    for i in range(n):
        for j in range(n):
            if j == i-1 or j == i+1:
                m[i][j] = 1
            elif j == i:
                m[i][j] = -2
            else:
                m[i][j] = 0
    return m

#Jacobi
def jacobi(A,b,atol,n):
    x = np.zeros(n,float)
    R = np.zeros(n,float)
    x_novo = np.zeros(n)
    r = []
    k = 0
    while True:
        for i in range(n):
            s = 0.0
            for j in range(n):
                s = s + A[i,j]*x[j]
            R[i] = (1.0/A[i,i])*(b[i] - s)
            x_novo[i] = x[i] + R[i]
        r.append(max(R))
        k+=1
        x = np.copy(x_novo)
        if np.sqrt(np.dot(R,R)) < atol:
            break
    return k, np.array(r)
#Gauss Seidel
def gauss(A,b,atol,n):
    x = np.zeros(n,float)
    R = np.zeros(n,float)
    r = []
    k = 0
```

```
    while True:
        for i in range(n):
            s = 0.0
            for j in range(n):
                s = s + A[i,j]*x[j]
            R[i] = (1.0/A[i,i])*(b[i]-s)
            x[i] = x[i] + R[i]
        r.append(max(R))
        k+=1

        if np.sqrt(np.dot(R,R)) < atol:
            break

    return k, np.array(r)

#SOR
def sor(A,b,atol,n):
    x = np.zeros(n,float)
    R = np.zeros(n,float)
    w = 1.5 #alterar w para maxima convergencia
    r = []
    k = 0
    while True:
        for i in range(n):
            s = 0.0
            for j in range(n):
                s = s + A[i,j]*x[j]
            R[i] = (1.0/A[i,i])*(b[i]-s)
            x[i] = x[i] + w*R[i]
        r.append(max(R))
        k+=1
        if np.sqrt(np.dot(R,R)) < atol:
            break

    return k, np.array(r)

#metodo gradiente conjugado
def gradconj(A,b,atol,n):
    k = 0
    r = []
```

```
    x = np.zeros(n,float)
    R = np.zeros(n,float)
    aux = np.dot(A,x)
    for i in range(n):
        R[i] = b[i] - aux[i]
    p = np.copy(R)
    while True:
        s = np.matmul(A,p)
        a = (np.dot(R,R))/(np.dot(p,s))
        R_aux = np.copy(R)
        for i in range(n):
            x[i] = x[i] + a*p[i]
            R[i] = R[i] - a*(s[i])
        r.append(max(R))
        k+=1

        if np.sqrt(np.dot(R,R)) < atol:
            break
        b = (np.dot(R,R))/(np.dot(R_aux,R_aux))
        for i in range(n):
            p[i] = R[i] + b*p[i]

    return k, np.array(r)

atol = 1e-7
n = 20

r_jacobi = np.zeros(n-1,int)
r_gauss = np.zeros(n-1,int)
r_sor = np.zeros(n-1,int)
r_gradconj = np.zeros(n-1,int)
i = 20
A = mat_n(i)
b = np.zeros(i)
b[i-1] = -1
#k_jacobi, r_jacobi = jacobi(A,b,atol,i)
#k_gauss, r_gauss = gauss(A,b,atol,i)
#k_sor, r_sor = sor(A,b,atol,i)
k_gradconj, r_gradconj = gradconj(A,b,atol,i)
```

```
#k_jacobi = np.linspace(0,k_jacobi-1,k_jacobi)
#k_gauss = np.linspace(0,k_gauss-1,k_gauss)
#k_sor = np.linspace(0,k_sor-1,k_sor)
k_gradconj = np.linspace(0,k_gradconj-1,k_gradconj)

#plt.plot(k_jacobi,r_jacobi)
#plt.plot(k_gauss,r_gauss)
#plt.plot(k_sor,r_sor)
plt.plot(k_gradconj,r_gradconj)
plt.show()
```

Referências

[Ros22] Adriano Possebon Rosa. *Equação do Calor e Diferenças Finitas.* https://aprender3.unb.br/pluginfile.php/2182438/mod_resource/content/6/$MNT_EqdoCal$ 2022.