Roteiro para o Trabalho Numérico de TCM: Solução da Equação do Calor 1D Transiente

Disciplina: Transporte de Calor e Massa
Professor: Adriano Possebon Rosa

1 Introdução

O objetivo neste Trabalho Numérico é determinar a temperatura em cada ponto de uma barra ao longo do tempo. Para isso, considere uma barra de $1\,m$ de comprimento, como mostra a figura (1). Queremos determinar a temperatura $T(x)$, que é uma função, para cada tempo, já que essa temperatura muda com o tempo. Assim, temos uma temperatura que depende do espaço e do tempo: $T(x,t)$.

Figura 1: Barra unidimensional.

A equação que nos dá o comportamento da temperatura é uma equação diferencial parcial homogênea linear de segunda ordem, conhecida como equação do calor. Em nosso curso de TCM nós deduzimos essa equação a partir da primeira lei da termodinâmica, obtendo

$$\rho c \frac{\partial T(x,t)}{\partial t} = k \frac{\partial^2 T(x,t)}{\partial x^2} \ . \tag{1}$$

Na equação (1), ρ é a densidade da barra, c é o calor específico e k é a condutividade térmica. A temperatura no instante de tempo t e na posição x é indicada por $T(x,t)$.A equação (1) pode ser reescrita como:

$$\frac{\partial T(x,t)}{\partial t} = \alpha \frac{\partial^2 T(x,t)}{\partial x^2} \ . \tag{2}$$

Na equação (2), $\alpha = k/\rho c$ é denominado difusividade térmica e representa uma razão entre a condução de calor e a capacidade de armazenamento de energia da placa.

Precisamos de uma condição inicial (a temperatura em todos os pontos no instante $t = 0$) e precisamos também de duas condições de contorno (a temperatura ou a derivada da temperatura nas extremidades da barra).

Aqui neste roteiro vamos resolver um exemplo específico. No trabalho, você deve resolver 3 problemas parecidos e discutir os resultados. Vamos começar com o nosso exemplo aqui do roteiro.

2 Problema

Por conveniência, vamos considerar, a partir deste ponto, $\alpha = 1\,m^2/s$. Essa especificação não afeta a interpretação física/qualitativa dos resultados, mas facilita a implementação numérica e a discussão dos resultados. Além disso, não serão mais colocadas as unidades das grandezas. Então a barra tem comprimento 1 e a temperatura vai de 0 a 1.

Este é o exemplo que vamos resolver aqui no roteiro:

$$\frac{\partial T(x,t)}{\partial t} = \frac{\partial^2 T(x,t)}{\partial x^2}\,, \qquad t > 0\,, \qquad 0 < x < 1\,. \tag{3}$$

$$T(0,t) = 0 \qquad \text{e} \qquad T(1,t) = 1, \qquad t \geq 0 \tag{4}$$

$$T(x,0) = 0, \qquad 0 < x < 1 \tag{5}$$

Qual é o significado dessas equações? A equação (3) é a equação do calor, que é equação diferencial governante do problema. Os princípios físicos por trás dessa equação são a primeira lei da termodinâmica (conservação de energia) e a lei de Fourier. **Note que ela é válida para $t > 0$ (no instante inicial ela não é válida) e para $0 < x < 1$ (só para o interior da barra).** As condições para os extremos da barra ($x = 0$ e $x = 1$) são dadas pela equação (4) e a condição no instante inicial ($t = 0$) é dada pela equação (5).

Na equação (4) temos que a temperatura na extremidade esquerda da barra é sempre 0 e na extremidade direita é sempre 1, para qualquer tempo. **As temperaturas nos pontos 0 e 1 não mudam: elas já estão resolvidas e pronto.**

Na equação (5) nós temos a temperatura para os pontos do interior da barra no instante de tempo inicial ($t = 0$).

As equações (4) são chamadas de condições de contorno e a equação (5) é chamada de condição inicial. Todos os problemas de condução de calor unidimensional transiente são governados pela equação (3), mas cada problema tem suas condições de contorno e inicial específicas, o que faz com que cada solução seja única. Lembre-se de que resolver esse problema significa encontrar uma solução que satisfaça ao mesmo tempo a equação governante e as condições de contorno e inicial.

Como é o comportamento da temperatura nesse problema? A figura (2) mostra a temperatura em $t = 0$, que é a condição inicial. Note que em todos os pontos a temperatura é igual a zero, exceto em $x = 1$, para o qual $T = 1$. A temperatura em $x = 0$ e em $x = 1$ não vai mudar ao longo do tempo, devido às condições de contorno.

A figura (3) mostra a temperatura em todos os pontos da barra para $t = 0,01$. Note que os pontos próximos da extremidade direita tiveram um aumento na temperatura, causado pela condução de calor. **Os pontos próximos à extremidade esquerda ainda não sentiram esse aumento.**

A figura (4) mostra a temperatura para $t = 0,05$. Aqui todos os pontos já sentiram a presença da extremidade direita aquecida. **Na figura (5), para $t = 0,5$, o perfil de temperatura é uma reta e não vai mais mudar com o tempo: chegamos no**

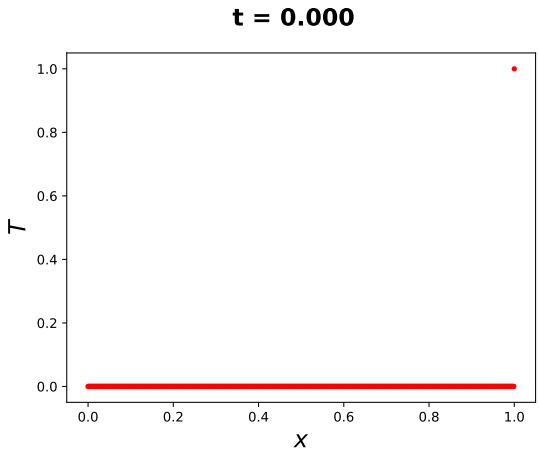

Figura 2: Temperatura em $t = 0$.

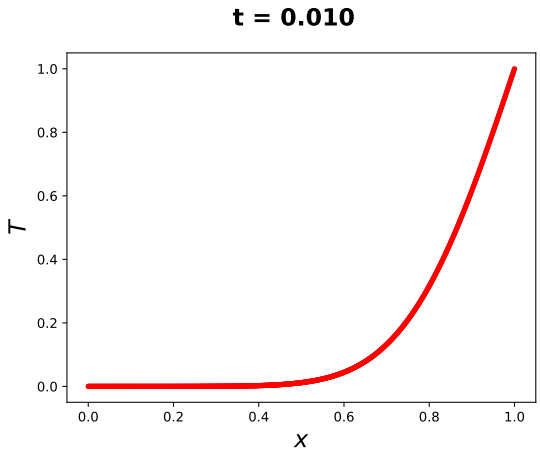

Figura 3: Temperatura em $t = 0,01$.

regime permanente. E por que não muda mais com o tempo? Porque, lembrando da equação governante

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

para que exista uma mudança no tempo é necessário que a derivada segunda da temperatura com relação a x seja diferente de zero. Mas se o gráfico é uma reta, então a derivada segunda é igual a zero, o que significa que a temperatura não muda mais com o tempo.

E como resolver esse problema, dado pelas equações (3), (4) e (5) ? Uma forma de resolver é utilizando um **método numérico** chamado de **Diferenças Finitas.**

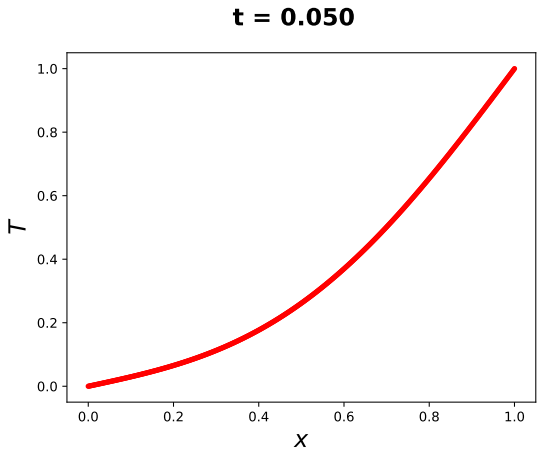

Figura 4: Temperatura em $t = 0,05$.

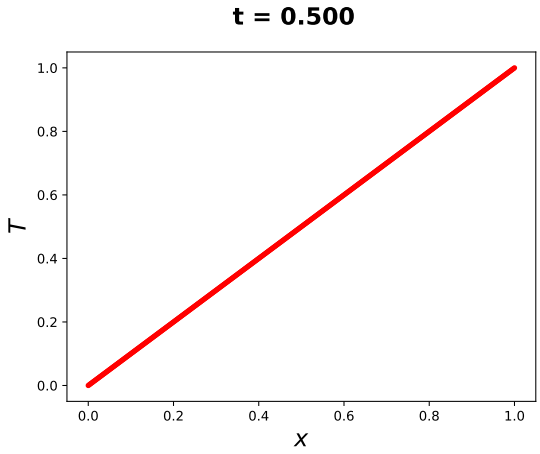

Figura 5: Temperatura em $t = 0,5$.

3 Método das Diferenças Finitas

A resolução de um problema usando o Método das Diferenças Finitas consiste de 4 etapas:

1. Dividir o domínio
2. Aproximar as derivadas usando séries de Taylor
3. Substituir na equação original
4. Resolver

Vamos ver essas etapas detalhadamente.

3.1 Dividindo o Domínio

No método das diferenças finitas, a temperatura é determinada apenas em pontos específicos do domínio e em certos instantes de tempo. Para o nosso problema da barra unidimensional, isso significa que a temperatura num ponto x_i e num tempo t_k é aproximada pela temperatura numérica (ou discreta):

$$T(x_i, t_k) \approx T_i^k \ . \tag{6}$$

Temos $x_i = i\Delta x$ e no instante de tempo $t_k = k\Delta t$. Aqui Δx é o incremento na variável espacial e Δt é o passo de tempo. Aqui i vai de 0 a $N+1$, onde N é o número de divisões do domínio. Já k vai de 0 até um valor qualquer definido. k e i são inteiros. Não temos mais a temperatura em qualquer ponto do domínio e em qualquer instante de tempo, mas apenas em posições e em tempos específicos (não temos mais infinitos pontos reais, mas apenas um número finito de pontos, que serão armazenados na memória do computador).

Para ilustrar essa discretização do domínio, vamos considerar aqui que a nossa barra está dividida em 5 partes ($N = 5$) iguais, ou seja, temos 6 pontos, como mostra a figura (6). (Note que a escolha $N = 5$ foi feita por mim, arbitrariamente. Veremos, mais a frente, que quanto maior N mais correta é a solução, porém maior é o custo computacional).

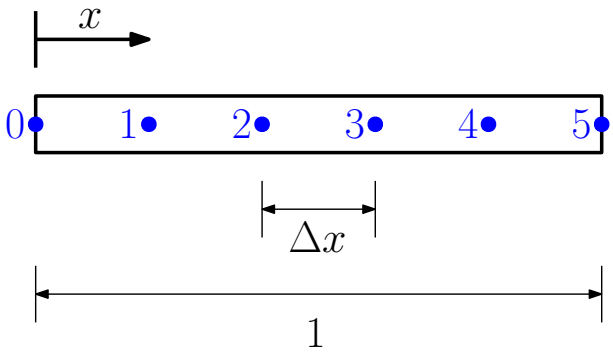

Figura 6: Barra unidimensional.

Temos os pontos 0, 1, 2, 3, 4 e 5. Esses são os nomes dos pontos, e não suas posições (x vai de zero a 1 somente). Esses pontos estão afastados por uma distância $\Delta x = 0,2$ entre si (a barra tem comprimento 1, então $\Delta x = 1/N = 1/5 = 0,2$). A temperatura é calculada apenas nesses pontos. Assim, a temperatura no ponto 0 é T_o, no ponto 1 é T_1, no ponto 2 é T_2 e assim por diante. De maneira geral, a temperatura no ponto i é T_i. O

ponto $i+1$ é o vizinho da direita do ponto i e o ponto $i-1$ é o vizinho da esquerda do ponto i. Assim, T_{i+1} é a temperatura do ponto que está à direita do ponto i e T_{i-1} é a temperatura do ponto que está à esquerda do ponto i.

Mas além de mudar no espaço, a temperatura também muda no tempo. O intervalo entre duas medições no tempo será Δt. Assim, vamos começar com $t = 0$, que é $0 \times \Delta t$. Depois vamos para o tempo $t_1 = 1 \times \Delta t$. Em seguida para $t_2 = 2 \times \Delta t$, e assim por diante. De maneira geral, temos $t_k = k \times \Delta t$. Se o instante atual é representado por k, o instante futuro próximo é dado por $k+1$.

A temperatura em um ponto i em um dado passo de tempo k é representada por T_i^k. Vamos considerar, por exemplo, $\Delta t = 0,01$. Então a temperatura em $x = 0,4$ e em $t = 0,02$ é dada por T_2^2. A temperatura em $x = 0,8$ e em $t = 0,13$ é T_4^{13}. Acho que deu pra entender a ideia. (Note, novamente, que eu escolhi $\Delta t = 0,01$ arbitrariamente. Na verdade, nem tanto. Para dar certo, temos que ter $\Delta t < 0,5\Delta x^2$, devido à condição de estabilidade. Além disso, quanto menor Δt melhor é a solução.)

Para o nosso problema, que é dado pelas equações (3), (4) e (5), temos que a condição inicial nos informa que todos os pontos do interior da barra têm temperatura igual a zero em $t = 0$. Para a nossa solução numérica, isso significa que

$$T_1^0 = 0 \qquad T_2^0 = 0 \qquad T_3^0 = 0 \qquad T_4^0 = 0 \ .$$

Por outro lado, as condições de contorno nos informam que a temperatura no ponto da extremidade esquerda é igual a zero sempre e a temperatura na extremidade direita é igual a 1 sempre. Isso significa que a temperatura do ponto 0 será sempre 0 e a temperatura do ponto 5 será sempre 1. Traduzindo para o nosso caso, temos:

$$T_o^o = 0 \qquad T_5^o = 1$$
$$T_o^1 = 0 \qquad T_5^1 = 1$$
$$T_o^2 = 0 \qquad T_5^2 = 1$$
$$\ldots\ldots\ldots$$

Isso significa que os pontos 0 e 5 já estão resolvidos. Só precisamos resolver os pontos 1, 2, 3 e 4.

Para o tempo $t = 0$ já está tudo resolvido. **Pergunta: como calcular os valores de T no próximo passo de tempo?** Ou seja, como calcular T_1^1, T_2^1, T_3^1 e T_4^1? Afinal é esse o nosso interesse, determinar o comportamento da temperatura ao longo do tempo. Nós conhecemos apenas valores em $k = 0$. Para determinar a temperatura no futuro vamos usar a equação governante.

3.2 Aproximando as Derivadas

Aí vem a segunda parte do método de diferenças finitas: vamos transformar a equação diferencial governante em várias equações algébricas. Para isso, vamos aproximar as derivadas da equação original utilizando séries de Taylor. Na equação original temos uma derivada primeira no tempo

$$\frac{\partial T}{\partial t}$$

e uma derivada segunda no espaço

$$\frac{\partial^2 T}{\partial x^2} \ .$$

Atenção: nessa subseção 3.2 são apresentados alguns conceitos mais teóricos. Se quiser pode ir direto para a subseção 3.3.

Considere a expansão em série de Taylor para a temperatura em $x_i + \Delta x$ em torno de x_i (em um tempo t_k qualquer fixo):

$$T(x_i + \Delta x, t_k) = T(x_i, t_k) + \Delta x \frac{\partial T}{\partial x}\bigg|_{x_i,t_k} + \frac{\Delta x^2}{2}\frac{\partial^2 T}{\partial x^2}\bigg|_{x_i,t_k} + \frac{\Delta x^3}{3!}\frac{\partial^3 T}{\partial x^3}\bigg|_{x_i,t_k} + \cdots \quad (7)$$

Mas $T(x_i + \Delta x, t_k) = T^k_{i+1}$ e $T(x_i, t_k) = T^k_i$. Assim, reescrevendo essa equação temos:

$$T^k_{i+1} = T^k_i + \Delta x \frac{\partial T}{\partial x}\bigg|_{x_i,t_k} + \frac{\Delta x^2}{2}\frac{\partial^2 T}{\partial x^2}\bigg|_{x_i,t_k} + \frac{\Delta x^3}{3!}\frac{\partial^3 T}{\partial x^3}\bigg|_{x_i,t_k} + \cdots \quad (8)$$

Isolando a primeira derivada de T com relação a x temos:

$$\frac{\partial T}{\partial x}\bigg|_{x_i,t_k} = \frac{T^k_{i+1} - T^k_i}{\Delta x} - \frac{\Delta x}{2}\frac{\partial^2 T}{\partial x^2}\bigg|_{x_i,t_k} - \frac{\Delta x^2}{3!}\frac{\partial^3 T}{\partial x^3}\bigg|_{x_i,t_k} + \cdots \quad (9)$$

Podemos escrever:

$$\frac{\partial T}{\partial x}\bigg|_{x_i,t_k} = \frac{T^k_{i+1} - T^k_i}{\Delta x} + R(\Delta x) \quad (10)$$

Aqui, $R(\Delta x)$ é uma função $\mathcal{O}(\Delta x)$, ou seja, ordem Δx. Se o último termo na equação (10) é truncado ou desprezado, temos então uma aproximação para a primeira derivada em x_i:

$$\frac{\partial T}{\partial x}\bigg|_{x_i,t_k} \approx \frac{T^k_{i+1} - T^k_i}{\Delta x} \ . \quad (11)$$

Essa é a aproximação da primeira derivada de uma função $T(x,t)$ no ponto x_i e no tempo t_k usando diferenças finitas. **Transformamos uma derivada espacial em uma equação algébrica, que usa pontos vizinhos do domínio discreto numérico.** Essa aproximação é para frente (pois usa um ponto que está na frente de x_i) e de primeira ordem (o erro é proporcional a Δx). Existem outras aproximações.

Considere novamente a série de Taylor em torno de x_i, mas agora para a função T em x_{i-1}:

$$T^k_{i-1} = T^k_i - \Delta x \frac{\partial T}{\partial x}\bigg|_{x_i,t_k} + \frac{\Delta x^2}{2}\frac{\partial^2 T}{\partial x^2}\bigg|_{x_i,t_k} - \frac{\Delta x^3}{3!}\frac{\partial^3 T}{\partial x^3}\bigg|_{x_i,t_k} + \cdots \quad (12)$$

Da equação (12), podemos aproximar:

$$\frac{\partial T}{\partial x}\bigg|_{x_i,t_k} \approx \frac{T^k_i - T^k_{i-1}}{\Delta x} \ . \quad (13)$$

Essa também é uma aproximação da primeira derivada espacial de $T(x,t)$ em x_i, mas agora usando um ponto para trás de i. Essa aproximação também é de ordem Δx.

Esses são exemplos de aproximações para a primeira derivada espacial. **Mas nós precisamos da segunda derivada espacial.** Para a segunda derivada, considere a soma das equações (8) e (12):

$$T^k_{i+1} + T^k_{i-1} = 2T^k_i + \Delta x \frac{\partial T}{\partial x}\bigg|_{x_i,t_k} - \Delta x \frac{\partial T}{\partial x}\bigg|_{x_i,t_k} + \frac{\Delta x^2}{2}\frac{\partial^2 T}{\partial x^2}\bigg|_{x_i,t_k} + \frac{\Delta x^2}{2}\frac{\partial^2 T}{\partial x^2}\bigg|_{x_i,t_k} + \\ + \frac{\Delta x^3}{6}\frac{\partial^3 T}{\partial x^3}\bigg|_{x_i,t_k} - \frac{\Delta x^3}{6}\frac{\partial^3 T}{\partial x^3}\bigg|_{x_i,t_k} + \frac{\Delta x^4}{24}\frac{\partial^4 T}{\partial x^4}\bigg|_{x_i,t_k} + \frac{\Delta x^4}{24}\frac{\partial^4 T}{\partial x^4}\bigg|_{x_i,t_k} + \cdots \quad (14)$$

Os termos com derivadas ímpares se cancelam (isso é fundamental, pois a primeira derivada desaparece da equação):

$$T_{i+1}^k + T_{i-1}^k = 2T_i^k + \Delta x \left.\frac{\partial T}{\partial x}\right|_{x_i,t_k} - \Delta x \left.\frac{\partial T}{\partial x}\right|_{x_i,t_k} + \frac{\Delta x^2}{2} \left.\frac{\partial^2 T}{\partial x^2}\right|_{x_i,t_k} + \frac{\Delta x^2}{2} \left.\frac{\partial^2 T}{\partial x^2}\right|_{x_i,t_k} + \\ + \frac{\Delta x^3}{6} \left.\frac{\partial^3 T}{\partial x^3}\right|_{x_i,t_k} - \frac{\Delta x^3}{6} \left.\frac{\partial^3 T}{\partial x^3}\right|_{x_i,t_k} + \frac{\Delta x^4}{24} \left.\frac{\partial^4 T}{\partial x^4}\right|_{x_i,t_k} + \frac{\Delta x^4}{24} \left.\frac{\partial^4 T}{\partial x^4}\right|_{x_i,t_k} + \cdots \tag{15}$$

Isolando a segunda derivada:

$$\left.\frac{\partial^2 T}{\partial x^2}\right|_{x_i,t_k} = \frac{T_{i+1}^k - 2T_i^k + T_{i-1}^k}{\Delta x^2} + R(\Delta x^2) \ . \tag{16}$$

Truncando a função R, resulta na aproximação usando diferenças finitas centradas da segunda derivada da função $T(x)$ no ponto x_i:

$$\left.\frac{\partial^2 T}{\partial x^2}\right|_{x_i,t_k} \approx \frac{T_{i+1}^k - 2T_i^k + T_{i-1}^k}{\Delta x^2} \ . \tag{17}$$

Note que essa aproximação é de segunda ordem, ou seja, estamos cometendo um erro que é proporcional a Δx^2. Nós não sabemos o valor do erro, mas sabemos que esse erro diminui 4 vezes quando o Δx passa a ser a metade.

A equação (17) é uma parte do que a gente queria: uma aproximação para a derivada espacial segunda. Falta uma aproximação para a derivada primeira no tempo.

Considere a expansão em série de Taylor para a temperatura em $t_k + \Delta t$ em torno de t_k (em um ponto x_i qualquer fixo):

$$T(x_i, t_k + \Delta t) = T(x_i, t_k) + \Delta t \left.\frac{\partial T}{\partial t}\right|_{x_i,t_k} + \frac{\Delta t^2}{2} \left.\frac{\partial^2 T}{\partial t^2}\right|_{x_i,t_k} + \frac{\Delta t^3}{3!} \left.\frac{\partial^3 T}{\partial t^3}\right|_{x_i,t_k} + \cdots \tag{18}$$

Mas $T(x_i, t_k + \Delta t) = T_i^{k+1}$ e $T(x_i, t_k) = T_i^k$. Assim, reescrevendo:

$$T_i^{k+1} = T_i^k + \Delta t \left.\frac{\partial T}{\partial t}\right|_{x_i,t_k} + \frac{\Delta t^2}{2} \left.\frac{\partial^2 T}{\partial t^2}\right|_{x_i,t_k} + \frac{\Delta t^3}{3!} \left.\frac{\partial^3 T}{\partial t^3}\right|_{x_i,t_k} + \cdots \tag{19}$$

Isolando a primeira derivada de T com relação a t temos:

$$\left.\frac{\partial T}{\partial t}\right|_{x_i,t_k} = \frac{T_i^{k+1} - T_i^k}{\Delta t} - \frac{\Delta t}{2} \left.\frac{\partial^2 T}{\partial t^2}\right|_{x_i,t_k} - \frac{\Delta t^2}{3!} \left.\frac{\partial^3 T}{\partial t^3}\right|_{x_i,t_k} + \cdots \tag{20}$$

Podemos escrever:

$$\left.\frac{\partial T}{\partial t}\right|_{x_i,t_k} = \frac{T_i^{k+1} - T_i^k}{\Delta t} + R(\Delta t) \tag{21}$$

Aqui, $R(\Delta t)$ é uma função $\mathcal{O}(\Delta t)$, ou seja, ordem Δt. Se o último termo na equação é truncado ou desprezado, temos então uma aproximação para a primeira derivada no ponto x_i no tempo t_k:

$$\left.\frac{\partial T}{\partial t}\right|_{x_i,t_k} \approx \frac{T_i^{k+1} - T_i^k}{\Delta t} \ . \tag{22}$$

Essa é a aproximação da primeira derivada temporal de uma função $T(x,t)$ no ponto x_i e no tempo t_k usando diferenças finitas. **Transformamos uma derivada temporal em uma equação algébrica, que usa um ponto no futuro e um ponto no presente.**

As equações (17) e (22) são as aproximações das derivadas que queremos. Vamos substituí-las na equação diferencial parcial original.

3.3 Substituição das Aproximações na Equação Original

A equação diferencial parcial original é dada por

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} \tag{23}$$

Vamos aproximar essa equação em um dado ponto e em um dado tempo usando as aproximações dadas pelas equações (17) e (22). Disso resulta:

$$\frac{T_i^{k+1} - T_i^k}{\Delta t} = \frac{T_{i+1}^k - 2T_i^k + T_{i-1}^k}{\Delta x^2} \ . \tag{24}$$

Essa é a nossa Equação de Diferenças Finitas. Para cada ponto e em cada instante de tempo temos uma equação algébrica. **Isso significa que transformamos uma única equação diferencial parcial em várias equações algébricas.**

Com as condições de contorno e a condição inicial dadas é possível calcular a nova temperatura T_i^{k+1} em todos os pontos i e, com isso, fazer a evolução temporal da temperatura, com um passo de cada vez. Para o nosso problema exemplo, vamos ver como se faz isso.

3.4 Resolvendo

A nossa equação de diferenças é a equação (24). Note que os valores no tempo k são conhecidos, já que é o tempo presente. Não conhecemos os valores em $k+1$, que é o tempo futuro. Podemos isolar T_i^{k+1} na equação, resultando em:

$$T_i^{k+1} = T_i^k + \frac{\Delta t}{\Delta x^2}\left(T_{i+1}^k - 2T_i^k + T_{i-1}^k\right) \ . \tag{25}$$

Vamos voltar agora ao nosso problema particular, o exemplo que estamos resolvendo neste roteiro. Temos $\Delta x = 0,2$ e $\Delta t = 0,01$, que resulta em $\Delta t/\Delta x^2 = 0,25$. Para $k = 0$ (ver figura 7),

$$T_0^0 = 0 \qquad T_1^0 = 0 \qquad T_2^0 = 0 \qquad T_3^0 = 0 \qquad T_4^0 = 0 \qquad T_5^0 = 1 \ .$$

Com esses valores, vamos usar a equação (25) com $k = 0$ e com i variando de 1 a 4:

$$T_1^1 = T_1^0 + 0,25\left(T_2^0 - 2T_1^0 + T_0^0\right) = 0 + 0,25(0 - 2\times 0 + 0) = 0 \ . \tag{26}$$

$$T_2^1 = T_2^0 + 0,25\left(T_3^0 - 2T_2^0 + T_1^0\right) = 0 + 0,25(0 - 2\times 0 + 0) = 0 \ . \tag{27}$$

$$T_3^1 = T_3^0 + 0,25\left(T_4^0 - 2T_3^0 + T_2^0\right) = 0 + 0,25(0 - 2\times 0 + 0) = 0 \ . \tag{28}$$

$$T_4^1 = T_4^0 + 0,25\left(T_5^0 - 2T_4^0 + T_3^0\right) = 0 + 0,25(1 - 2\times 0 + 0) = 0,25 \ . \tag{29}$$

Assim, no tempo $0,01$, que corresponde a $k = 1$, temos (ver figura 8)

$$T_0^1 = 0 \qquad T_1^1 = 0 \qquad T_2^1 = 0 \qquad T_3^1 = 0 \qquad T_4^1 = 0,25 \qquad T_5^1 = 1 \ .$$

Com isso avançamos 1 passo de tempo. Agora é só continuar avançando no tempo. Duas observações:

- para avançar um passo no futuro, precisamos apenas da solução no tempo presente;
- para um dado tempo, primeiro calculamos a temperatura em todos os pontos da barra, e somente depois é possível avançar no tempo. Não é possível avançar no tempo apenas em um ponto e depois voltar no tempo e calcular para um outro ponto (fisicamente também não faz sentido).

Vamos usar agora a equação (25) com $k = 1$ e com i indo de 1 a 4:

$$T_1^2 = T_1^1 + 0,25\left(T_2^1 - 2T_1^1 + T_0^1\right) = 0 + 0,25(0 - 2 \times 0 + 0) = 0 \ . \tag{30}$$

$$T_2^2 = T_2^1 + 0,25\left(T_3^1 - 2T_2^1 + T_1^1\right) = 0 + 0,25(0 - 2 \times 0 + 0) = 0 \ . \tag{31}$$

$$T_3^2 = T_3^1 + 0,25\left(T_4^1 - 2T_3^1 + T_2^1\right) = 0 + 0,25(0,25 - 2 \times 0 + 0) = 0,0625 \ . \tag{32}$$

$$T_4^2 = T_4^1 + 0,25\left(T_5^1 - 2T_4^1 + T_3^1\right) = 0,25 + 0,25(1 - 2 \times 0,25 + 0) = 0,375 \ . \tag{33}$$

Assim, no tempo $0,02$, que corresponde a $k = 2$, temos (ver figura 9)

$$T_0^2 = 0 \qquad T_1^2 = 0 \qquad T_2^2 = 0 \qquad T_3^2 = 0,0625 \qquad T_4^2 = 0,375 \qquad T_5^2 = 1 \ .$$

E podemos continuar essas contas no tempo, para encontrar a temperatura em tempos futuros. Note que x é limitado, entre 0 e 1. Mas o tempo vai de zero até o valor que você quiser.

Se pensarmos na temperatura como um vetor, temos, para cada tempo:

$$T^o = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]$$

$$T^1 = [0 \quad 0 \quad 0 \quad 0 \quad 0,25 \quad 1]$$

$$T^2 = [0 \quad 0 \quad 0 \quad 0,0625 \quad 0,375 \quad 1]$$

.....

Essa ideia de vetor será útil na programação.

Observação: os valores de temperatura são calculados apenas nos 6 pontos escolhidos, como mostram as figuras (7), (8) e (9). Você pode traçar uma curva unindo esses pontos, pra ficar mais parecido com um gráfico de verdade, como mostra a figura (10). Mas é importante ficar claro que os valores calculados são apenas os 6 pontos e os valores entre os pontos são apenas suposições.

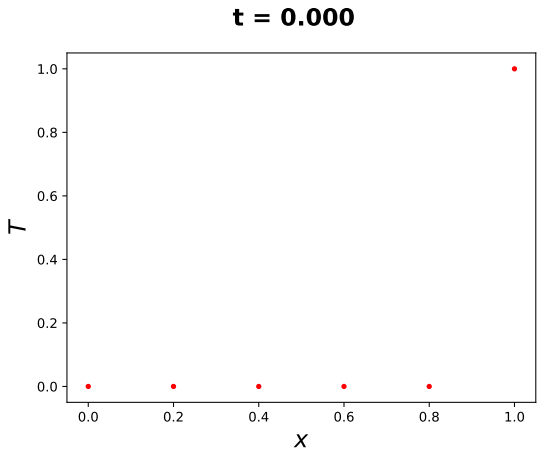

Figura 7: Temperatura para $k = 0$.

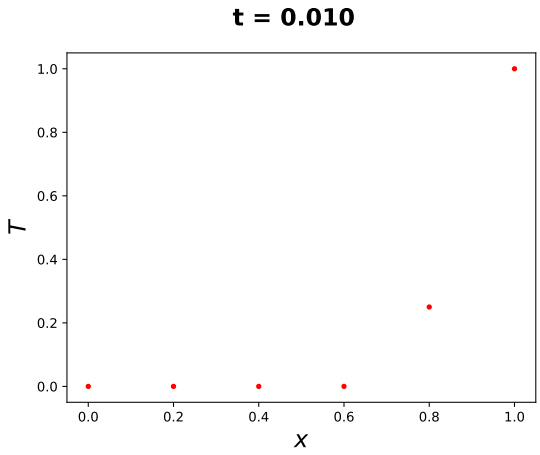

Figura 8: Temperatura para $k = 1$.

Ok, tudo certo. Mas se a gente consegue calcular na mão, como fizemos até aqui, **por que usar um computador?**

Bom, como vimos na parte de série de Taylor, a nossa aproximação tem um erro que é proporcional a Δx^2 e a Δt. Então quanto menores Δx e Δt, menor é o nosso erro numérico, e mais a nossa solução se aproxima da solução verdadeira. Por isso precisamos colocar muitos pontos no domínio. Nesse exemplo, nós utilizamos $N = 5$. Mas e se $N = 1000$? E se eu quiser a temperatura em $t = 10$? Vou ter que fazer essas contas 1000 vezes. Aí se torna muito complicado realizar esses cálculos à mão. **Melhor deixar o computador fazer essas contas pra gente.**

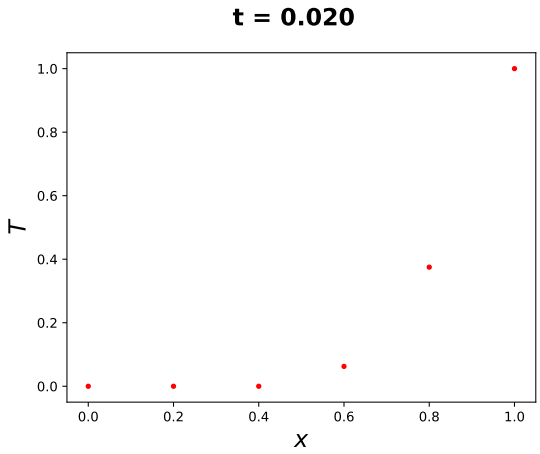

Figura 9: Temperatura para $k = 2$.

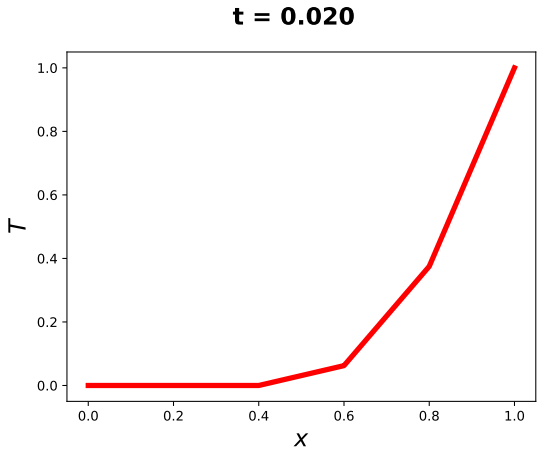

Figura 10: Temperatura para $k = 2$.

4 Programando com Python

4.1 Por que o Python?

Para resolver esse problema numericamente vamos precisar de uma linguagem de programação. Existem várias e todas têm seus pontos positivos e negativos. Alguns exemplos de linguagens:

C++ C Matlab R Fortran

Python Lua Julia Ruby Java

Por que o Python? É gratuito; open-source; fácil de aprender; possui um código mais limpo e mais fácil de ler; pode ser usado em diversas aplicações (sites, análise de dados, inteligência artificial, etc); é muito utilizado pela indústria; muitos manuais e tutoriais em português.

Atenção: você pode usar a linguagem de programação que você quiser. Não é obrigatório usar o Python.

Instalação: não sei como instalar o Python. Vou deixar essa parte com você. Procure Anaconda/Spyder, acho que é o jeito mais fácil de instalar e trabalhar com o python. Se nada der certo, procure 'python online' no Google e programe no navegador mesmo.

A figura (11) mostra um exemplo com o Spyder.

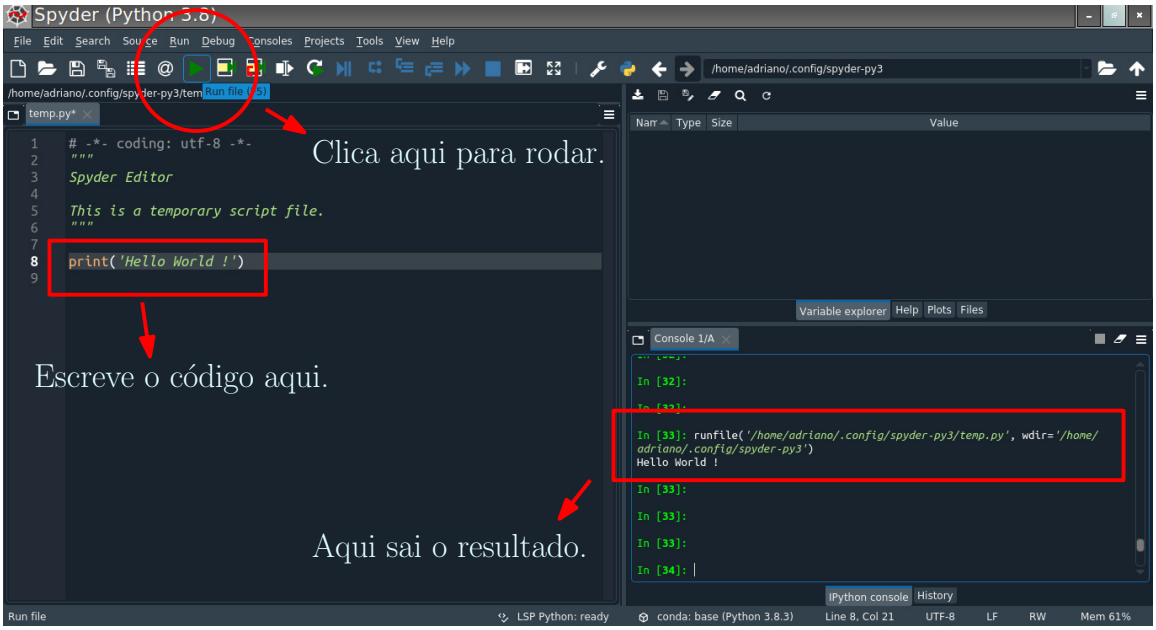

Figura 11: Spyder.

4.2 Comandos Básicos

O objetivo aqui é apresentar alguns comandos que vamos utilizar para resolver o nosso exemplo do roteiro. As explicações serão por meio de exemplos. **Os códigos serão dispostos em azul aqui ao longo do texto (cada bloco em azul é um código completo). Os resultados dos códigos serão colocados em vermelho. Atenção: é importante que você reproduza os códigos abaixo para ir se familiarizando**

com os comandos do python. Você tem que testar e sentir como a linguagem funciona. Não copie e cole os programas: escreva você mesmo!

Vamos começar com o famoso

```
print('Hello World !')
```

Essa função print() escreve na tela o que estiver entre parênteses. (Eu escrevi apenas 'Hello World !', esses traços horizontais entre as duas palavras e antes do ponto de exclamação é para indicar um espaço vazio.) O resultado seria então simplesmente

```
Hello World !
```

Vamos usar aqui 3 tipos de variáveis do python. Temos os inteiros, os *float numbers* (que são números com casas decimais) e as *strings* (sequências de caracteres). Vamos começar com os inteiros. Você escreve a seguinte expressão no seu código:

```
a = 7
```

O que isso significa? Isso significa que o computador reservou uma dada quantidade de memória (*bits*) e chamou essa parte da memória de *a*. Aí, a essa parte da memória chamada de *a* foi associado o número 7. Note que esse código não vai ter nenhum resultado visível, porque não foi utilizada a função *print*.

Podemos fazer operações com essas variáveis. Temos soma +, subtração -, multiplicação * e divisão /. Considere:

```
a = 7
print(2*a)
```

O resultado é:

```
14
```

Agora:

```
a = 7
b = 3
print(a*b)
```

O resultado é:

```
21
```

Agora:

```
a = 7
b = 3
print(a+b, a-b, a*b, a/b)
```

```
10 4 21 2.3333333333333335
```

Note que o resultado a/b é um *float*. O python 3 transforma o resultado da divisão entre dois inteiros automaticamente em um *float* (o python 2 não faz isso, dando como resultado simplesmente 2).

Para definir um *float* basta colocar o ponto e o zero depois do número.

```
a = 7
b = 3.0
print(type(a))
print(type(b))
```

```
c = a + b
print(c)
print(type(c))
```

```
class 'int'
class 'float'
10.0
class 'float'
```

Note que a função *type* dá o tipo da variável. É sempre importante escrever a variável como inteiro ou *float* de acordo com a especificação do problema.

Um *string* é um conjunto de caracteres. Devemos escrever entre aspas (simples ou duplas). Exemplo:

```
a = 'Transporte'
b = ' de Calor '
c = ' e Massa'
print(a)
print(b)
print(c)
d = a + b + c
print(d)
print(type(a))
```

```
Transporte
 de Calor
 e Massa
Transporte de Calor e Massa
class 'str'
```

Qual seria o resultado desse código?

```
a = 7
a = a + 5
print(a)
```

Aqui temos uma soma recursiva. Note que $a = a + 5$, na programação, não indica uma equação matemática como estamos acostumados. O que queremos dizer é que vamos armazenar na posição a da memória o valor que já estava armazenado em a anteriormente (no caso 7) somado de 5. O resultado é:

```
12
```

Outros exemplos:

```
a = 16.0
a = a/2
a = a/2
print(a)
```

```
2.0
```

```
a = 16
print(a)
```

```
print(type(a))
a = a/2
print(a)
print(type(a))
```

```
16
class 'int'
8.0
class 'float'
```

Aqui é interessante notar que o python mudou o tipo da variável *a* automaticamente.

Podemos fazer comentários no código que não serão "lidos" pelo python, mas que podem ajudar muito na organização do código. Para isso usamos #. Exemplo:

```
b = 3  #Base do retangulo
h = 8  #Altura do retangulo

#A area do retangulo eh dada pela base vezes a altura

A = b*h
print(A)
```

```
24
```

Muitas vezes precisamos que um mesmo procedimento seja executado repetidas vezes. Para isso usamos os *loops*, ou laços de repetição. Em python temos o *for* e o *while*. Vejamos alguns exemplos.

```
for i in range(2,7):
    print(i)
```

```
2
3
4
5
6
```

Note o espaço antes de *print*, isso indica que essa função está dentro do *loop*.

Veja o exemplo abaixo: 'oi' está dentro do *loop*, mas 'adeus' está fora.

```
for i in range(2,7):
    print('oi')
print('adeus')
```

```
oi
oi
oi
oi
oi
adeus
```

Neste caso, o *i* é a variável de repetição e *range(2,7)* significa que o *i* vai variar de 2 a 6: sempre começa no número da esquerda e termina no número da direita menos 1. Para *loops* começando do zero podemos indicar apenas o valor final. Mais alguns exemplos.

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

```
for i in range(-3,2):
    print(i)
```

```
-3
-2
-1
0
1
```

Nesses 4 exemplos de *loop* o procedimento que foi executado repetidas vezes foi apenas um *print*. Podemos ter qualquer tipo de comando sendo repetido.

```
for i in range(0,4):
    a = 2*i
    print('Valor de a:', a)
```

```
Valor de a: 0
Valor de a: 2
Valor de a: 4
Valor de a: 6
```

```
for i in range(0,4):
    a = 2*i*i
    print('Valor de a:', a)
```

```
Valor de a: 0
Valor de a: 2
Valor de a: 8
Valor de a: 18
```

Note a diferença entre os dois códigos abaixo:

```
a = 0
for i in range(1,6):
    a = a+i
    print('Valor de a:', a)
```

```
Valor de a: 1
Valor de a: 3
Valor de a: 6
Valor de a: 10
Valor de a: 15
```

```
a = 0
for i in range(1,6):
    a = a+i
print('Valor de a:', a)
```

Valor de a: 15

O que mudou é que a função *print* estava **dentro** do *loop* no primeiro código, mas **fora** do *loop* no segundo. Por isso no segundo resultado o valor de a é mostrado apenas uma vez, depois que todos os passos do *loop* foram realizados. No código acima calculamos

$$a = \sum_{i=1}^{5} i \ .$$

Assim, podemos usar soma recursiva para calcular somatórias.

Os dois códigos abaixo produzem o mesmo resultado. Não preciso nem comentar qual é o mais eficiente.

```
a = 0
for i in range(1,6):
    a = a + i*i
print('Valor de a:', a)
```

Valor de a: 55

```
a = 0
a = a + 1*1
a = a + 2*2
a = a + 3*3
a = a + 4*4
a = a + 5*5
print('Valor de a:', a)
```

Valor de a: 55

Podemos ainda ter um *loop* dentro de outro *loop*. Para isso é necessário usar duas variáveis diferentes. Exemplo:

```
for i in range(3):
    for j in range(2):
        print(i,j)
```

```
0 0
0 1
1 0
1 1
2 0
2 1
```

A outra maneira de fazer um *loop* é com o comando *while*. Esse comando vai repetir as operações dentro do bloco enquanto uma dada condição for verdadeira. Exemplos:

```
a = 0
print('Valor de a:', a)
while a < 5:
    a = a + 1
    print('Valor de a:', a)
```

```
Valor de a: 0
Valor de a: 1
Valor de a: 2
Valor de a: 3
Valor de a: 4
Valor de a: 5
```

```
a = 0
print('Valor de a:', a)
while a <= 5:
    a = a + 1
    print('Valor de a:', a)
```

```
Valor de a: 0
Valor de a: 1
Valor de a: 2
Valor de a: 3
Valor de a: 4
Valor de a: 5
Valor de a: 6
```

O *loop* continua até que a condição não seja mais verdadeira. Quais seriam os resultados dos dois código abaixo?

```
a = 0
print('Valor de a:', a)
while a > 5:
    a = a + 1
    print('Valor de a:', a)
```

```
????
```

```
a = 0
print('Valor de a:', a)
while a < 5:
    print('Valor de a:', a)
```

```
????
```

Temos também os operadores condicionais *if*, *elif* e *else*. O que tem dentro de um bloco *if* só será executado se a condição inicial for obedecida. Condições auxiliares são dadas por *elif* e *else*. Melhor explicar por meio de exemplos:

```
x = 1.0
if x > 0.0:
    print('x eh maior que zero')
```

x eh maior que zero

```
x = -1.0
if x > 0.0:
    print('x eh maior que zero')
```

NÃO VAI APARECER NADA NA RESPOSTA

```
x = 0.0
if x > 0.0:
    print('x eh maior que zero')
```

NÃO VAI APARECER NADA NA RESPOSTA

```
x = 0.0
if x > 0.0:
    print('x eh maior que zero')
print('x eh igual a zero')
```

x eh igual a zero

```
x = 1.0
if x > 0.0:
    print('x eh positivo')
elif x == 0.0:
    print('x eh igual a zero')
else:
    print('x eh menor que zero')
```

x eh positivo

```
x = 0.0
if x > 0.0:
    print('x eh positivo')
elif x == 0.0:
    print('x eh igual a zero')
else:
    print('x eh menor que zero')
```

x eh igual a zero

```
x = -3.0
if x > 0.0:
    print('x eh positivo')
elif x == 0.0:
    print('x eh igual a zero')
else:
    print('x eh menor que zero')
```

x eh negativo

Existem também as listas em python. Listas são elementos que podem armazenar vários valores. Por exemplo:

```
a = [4, 19, 3, -75]
print(a)
```

```
[4, 19, 3, -75]
```

No caso do exemplo acima, *a* é a lista (é um vetor). Podemos acessar os elementos da lista:

```
a = [4, 19, 3, -75]
print(a[2])
```

```
3
```

A posição na lista começa sempre do zero, ou seja, o índice da lista começa sempre do zero.

```
a = [4, 19, 3, -75]
print(a[0])
print(a[1])
print(a[2])
print(a[3])
```

```
4
19
3
-75
```

O código abaixo vai gerar um erro, pois estamos tentando acessar um elemento que não existe.

```
a = [4, 19, 3, -75]
print(a[4])
```

```
ERRO!!!!
```

Podemos usar loops para acessar os elementos de uma lista e modificá-los. Mais alguns exemplos abaixo:

```
a = [4, 19, 3, -75]
for i in range (4):
    print(a[i])
```

```
4
19
3
-75
```

```
a = [0, 0, 0, 0]
for i in range (4):
    a[i] = 3*i
print(a)
```

```
[0, 3, 6, 9]
```

```
a = [5, 5, 5, 5]
print(a)
a[1] = 33
print(a)
a[0] = a[0] + a[1] + a[2]
print(a)
```

```
[5, 5, 5, 5]
[5, 33, 5, 5]
[43, 33, 5, 5]
```

Apesar de ser possível resolver o nosso problema com listas (de maneira bem limitada), nós vamos utilizar as matrizes da **biblioteca numpy**. O python possui várias funções intrínsecas, mas algumas devem ser importadas de bibliotecas específicas. Vamos usar duas bibliotecas aqui: **numpy e matplotlib.pyplot** . A primeira é para usar matrizes e a segunda é para plotar os gráficos. Para usar essas bibliotecas nós temos que importá-las no início do programa. Exemplo:

```
import numpy

a = numpy.pi
print(a)
b = numpy.sin(a/2)
print(b)
```

```
3.141592653589793
1.0
```

Pra usar uma função devemos escrever "biblioteca ponto função". Podemos dar um apelido para a biblioteca, para facilitar na hora de escrever as funções (para a numpy parece não fazer muita diferença, mas para a matplotlib.pyplot vai ajudar bastante):

```
import numpy as np

a = np.pi
print(a)
b = np.sin(a/2)
print(b)
```

```
3.141592653589793
1.0
```

Vamos criar uma matriz com a biblioteca numpy:

```
import numpy as np

a = np.array([19, -3, 42, np.pi])
print(a)
print('A resposta para a Vida, o Universo e Tudo mais eh', a[2])
```

```
[19. -3. 42. 3.14159265]
```

A resposta para a Vida, o Universo e Tudo o mais eh 42.0

Podemos criar matrizes com várias linhas e colunas.

```
import numpy as np

a = np.array([[19, -3, 4], [1.0, 7.0, 4.0], [-14.0, 2.0, 29.0]])
print(a)
print(a[1,2])
print(a[0,0])
print(a[2,2])
```

```
[[ 19. -3. 4.]
 [ 1. 7. 4.]
 [-14. 2. 29.]]
4.0
19.0
29.0
```

Podemos criar matrizes específicas, de uns, zeros ou sequências estabelecidas. Isso é muito importante quando estamos lidando com grandes problemas, pois não vamos ficar montando matrizes elemento por elemento, manualmente. Novamente, acho que a melhor maneira de explicar é com exemplos:

```
import numpy as np

a = np.zeros(5,float)
print(a)
b = np.zeros(5,int)
print(b)
c = np.zeros(7,float)
print(c)
d = np.ones(4,float)
print(d)
e = np.linspace(0.0,1.0,11)
print(e)
f = np.arange(1,10,2)
print(f)
```

```
[0. 0. 0. 0. 0.]
[0 0 0 0 0]
[0. 0. 0. 0. 0. 0. 0.]
[1. 1. 1. 1.]
[0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
[1 3 5 7 9]
```

Note como as funções *linspace* e *arange* funcionam. A primeira criou um vetor com 11 elementos distribuídos linearmente entre 0.0 e 1.0. A segunda criou um vetor entre 1 e 10 indo de 2 em 2. Essas funções são muito úteis.

Outra função útil é a *np.copy*, que cria uma nova matriz idêntica à matriz original, mas independente. Veja:

```
import numpy as np

a = np.array([4,4,4])
b = np.copy(a)
print(a)
print(b)

a[1] = 10
b[2] = 13
print(a)
print(b)
```

```
[4 4 4]
[4 4 4]
[ 4 10 4]
[ 4 4 13]
```

O *np.copy* é importante pois se usarmos apenas a igualdade, na verdade não estaremos criando uma nova matriz independente. Quando usamos a igualdade nesse caso, o que acontece é que o python dá dois nomes diferentes ao mesmo local na memória. Veja:

```
import numpy as np

a = np.array([4,4,4])
b = a
print(a)
print(b)

a[1] = 10
print(a)
print(b)
```

```
[4 4 4]
[4 4 4]
[ 4 10 4]
[ 4 10 4]
```

Nesse último exemplo nós mudamos uma componente da matriz a mas a matriz b também mudou, pois as duas na verdade são a mesma coisa. Por isso é importante usar o *np.copy* para evitar esse problema.

Excelente! Agora vamos plotar o nosso primeiro gráfico. usando a biblioteca matplotlib.pyplot . Para plotar um gráfico precisamos de dois vetores. No exemplo abaixo vamos plotar x e y:

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

x = np.array([1,2,3,4])
y = np.array([3,6,9,12])

plt.plot(x,y)
```

```
plt.show()
```

O RESULTADO É O GRÁFICO.

Note que nós usamos a biblioteca *matplotlib.pyplot* para plotar o gráfico. Demos a essa biblioteca o apelido *plt*. Então para chamar uma função dessa biblioteca, nós escrevemos "plt ponto função". A função *plt.plot* plota os dois vetores x e y e a função *plt.show* mostra esses dois vetores. Aqui pra mim, no Spyder, o resultado é mostrado na figura (12). Note que não teve nenhum output numérico, já que não tem nenhum *print* no código.

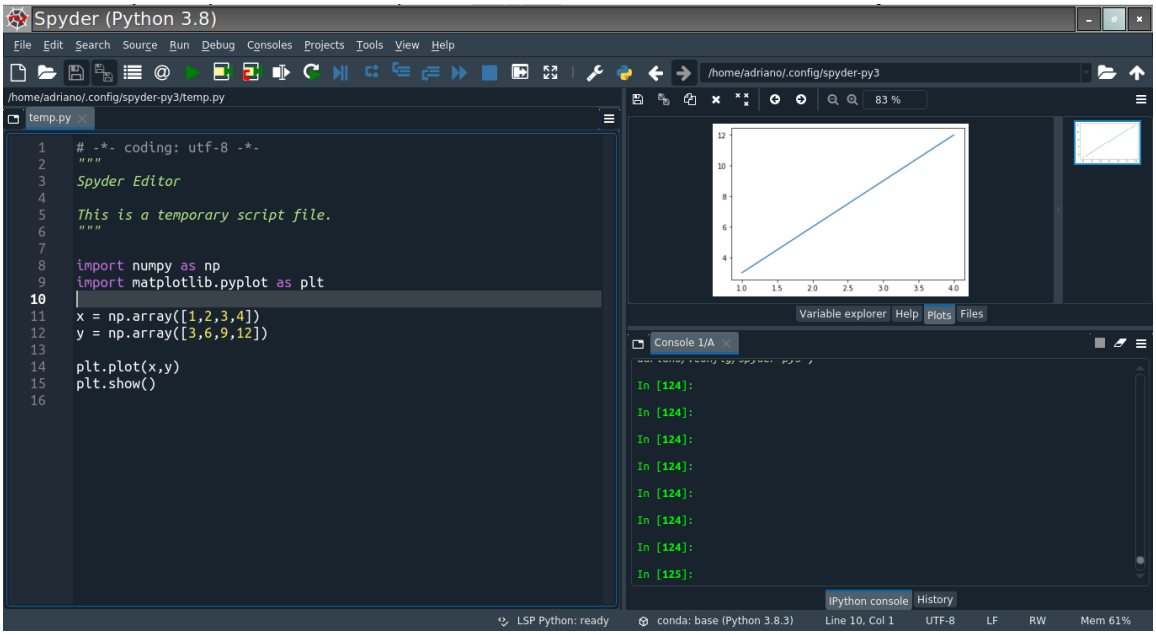

Figura 12: Spyder.

É possível salvar a figura externamente, para poder utilizá-la em um relatório, por exemplo. Veja o código abaixo:

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

x = np.array([1,2,3,4])
y = np.array([3,6,9,12])

plt.savefig('figura.pdf', format='pdf', dpi=1200,
 bbox_inches='tight')
plt.plot(x,y)
plt.show()
```

O RESULTADO É O GRÁFICO.

Nesse caso vai aparecer uma cópia da figura na pasta onde o programa está sendo executado. Vamos agora plotar uma função seno:

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

```
x = np.linspace(0.0,4.0*np.pi,200)
y = np.sin(x)

plt.plot(x,y)
plt.show()
```

O RESULTADO É O GRÁFICO.

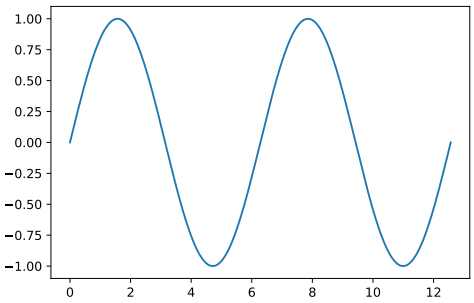

Figura 13: Seno

Bom, aí o que falta é trabalhar nos detalhes do gráfico. Podemos mudar a cor da linha, colocar pontos, colocar os títulos dos eixos e assim por diante. Vou deixar para você dar uma olhada no Google e ver as diversas opções. Aqui tem um exemplo:

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

x = np.linspace(0.0,4.0*np.pi,200)
y = np.sin(x)
z = np.cos(x)

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

fig.suptitle('Funcao Seno e Cosseno', fontsize=18, fontweight='bold')
ax.set_ylabel('$y$ e $z$', fontsize=18)
ax.set_xlabel('$x$', fontsize=18)

plt.plot(x, y, '-r', lw = 4)
plt.plot(x, z, '.b', lw = 4)

plt.savefig('figura.pdf', format='pdf', dpi=1200, bbox_inches='tight')

plt.show()
```

O RESULTADO É O GRÁFICO.

Concluindo: aí estão alguns comandos básicos do python. Como mencionado lá no começo dessa seção, essa é só a ponta do iceberg. Existem vários outros comandos e funções importantes no python. Além disso, as funções apresentadas aqui possuem outros

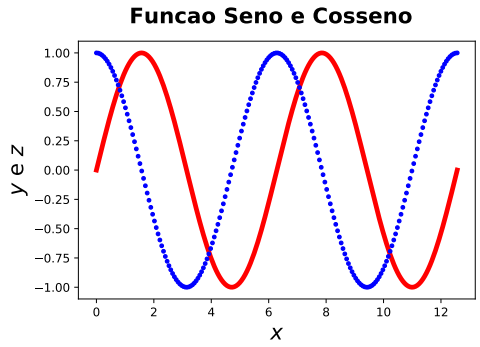

Figura 14: Seno e cosseno.

artifícios mais complexos. No entanto, para o que a gente deseja fazer aqui nesse trabalho, o que foi apresentado é o suficiente.

4.3 Programando o nosso Problema

Depois desse breve passeio por algumas funcionalidades do python, vamos voltar para o nosso problema.

Recapitulando, estamos resolvendo esse problema:

$$\frac{\partial T(x,t)}{\partial t} = \frac{\partial^2 T(x,t)}{\partial x^2} \ , \qquad t > 0 \ , \qquad 0 < x < 1 \ . \tag{34}$$

$$T(0,t) = 0 \qquad \text{e} \qquad T(1,t) = 1, \qquad t \geq 0 \tag{35}$$

$$T(x,0) = 0, \qquad 0 < x < 1 \tag{36}$$

Com o método das diferenças finitas, transformamos essa equação diferencial parcial em várias equações algébricas, uma para cada ponto do domínio e para cada instante de tempo. Dê uma olhada na seção 3. Chegamos na seguinte equação de diferenças:

$$\frac{T_i^{k+1} - T_i^k}{\Delta t} = \frac{T_{i+1}^k - 2T_i^k + T_{i-1}^k}{\Delta x^2} \ . \tag{37}$$

Ou:

$$T_i^{k+1} = T_i^k + \frac{\Delta t}{\Delta x^2}\left(T_{i+1}^k - 2T_i^k + T_{i-1}^k\right) \ . \tag{38}$$

No nosso problema temos $\Delta x = 0,2$ $(N = 5)$ e $\Delta t = 0,01$. Além disso, das condições inicial e de contorno resulta:

$$T_0^0 = 0 \qquad T_1^0 = 0 \qquad T_2^0 = 0 \qquad T_3^0 = 0 \qquad T_4^0 = 0 \qquad T_5^0 = 1 \ .$$

Ou:

$$T^o = [0.0 \qquad 0.0 \qquad 0.0 \qquad 0.0 \qquad 0.0 \qquad 1.0] \, .$$

Vamos calcular a temperatura do primeiro passo de tempo usando o python. Vamos começar com um programa bem simples e manual e depois vamos deixando ele mais eficiente e automatizado. Começando:

```
import numpy as np

delta_x = 0.2        #delta x
delta_t = 0.01       #delta t

#Agora vamos criar o vetor temperatura
#no instante atual

Temp = np.array([0.0,0.0,0.0,0.0,0.0,1.0])

#Vamos ver essa temperatura:

print('Temperatura inicial:', Temp)

#Queremos calcular a temperatura no proximo
#instante de tempo. Vamos chamar essa temperatura
#de Temp_nova. Vamos dizer por enquanto que ela eh
#igual a temperatura atual, Temp.

Temp_nova = np.copy(Temp)

#Agora vamos usar a equacao de diferencas para
#avancar cada ponto no tempo. Os pontos 0 e 5
#nao vao mudar, pois ja estao resvolvidos pela
#condicao de contorno.

#Para o ponto 1:

Temp_nova[1] = Temp[1] + (delta_t/(delta_x*delta_x))*(Temp[2]
                          - 2.0*Temp[1] + Temp[0])

#Para o ponto 2:

Temp_nova[2] = Temp[2] + (delta_t/(delta_x*delta_x))*(Temp[3]
                          - 2.0*Temp[2] + Temp[1])

#Para o ponto 3:

Temp_nova[3] = Temp[3] + (delta_t/(delta_x*delta_x))*(Temp[4]
                          - 2.0*Temp[3] + Temp[2])

#Para o ponto 4:

Temp_nova[4] = Temp[4] + (delta_t/(delta_x*delta_x))*(Temp[5]
                          - 2.0*Temp[4] + Temp[3])

#Pronto. Calculamos a nova temperatura em todos
#os pontos do interior do dominio. Como avancamos
```

```
#um passo de tempo, vamos atualizar a temperatura.

Temp = np.copy(Temp_nova)

#Vamos ver o resultado:

print('Temperatura final:', Temp)
```

Temperatura inicial: [0. 0. 0. 0. 0. 1.]
Temperatura final: [0. 0. 0. 0. 0.25 1.]

Esse é o resultado após 1 passo de tempo. Ou seja, essa é uma aproximação para a temperatura em todos os pontos da barra para $t = 0,01$. Esse resultado coincide com o que nós já tínhamos calculado lá na seção 3.4. Dá para melhorar esse código? Bom, primeiro a gente poderia dar um jeito de atualizar essas temperaturas mais rapidamente. Escrever manualmente uma equação para $Temp\_nova[1]$, $Temp\_nova[2]$, $\cdots$, não é uma boa ideia. Imagine se a gente tivesse 100 pontos. Vamos colocar isso dentro de um *loop*:

```
import numpy as np

delta_x = 0.2       #delta x
delta_t = 0.01      #delta t

#Agora vamos criar o vetor temperatura
#no instante atual

Temp = np.array([0.0,0.0,0.0,0.0,0.0,1.0])

#Vamos ver essa temperatura:

print('Temperatura inicial:', Temp)

#Queremos calcular a temperatura no proximo
#instante de tempo. Vamos chamar essa temperatura
#de Temp_nova. Vamos dizer por enquanto que ela eh
#igual a temperatura atual, Temp.

Temp_nova = np.copy(Temp)

#Agora vamos usar a equacao de diferencas para
#avancar cada ponto no tempo. Os pontos 0 e 5
#nao vao mudar, pois ja estao resvolvidos pela
#condicao de contorno.

#Para os pontos de 1 a 4 (esse eh o loop principal
#do programa, eh o coracao):

for i in range(1,5):
    Temp_nova[i] = Temp[i] + (delta_t/(delta_x*delta_x))*(Temp[i+1]
                           - 2.0*Temp[i] + Temp[i-1])
```

```
#Pronto. Calculamos a nova temperatura em todos
#os pontos do interior do dominio. Como avancamos
#um passo de tempo, vamos atualizar a temperatura.

Temp = np.copy(Temp_nova)

#Vamos ver o resultado:

print('Temperatura final:', Temp)
```

Temperatura inicial: [0. 0. 0. 0. 0. 1.]
Temperatura final: [0. 0. 0. 0. 0.25 1.]

Melhorou? Um pouco. E se eu quiser saber a temperatura mais no futuro? Por exemplo, depois de dois ou três passos de tempo? Eu teria que repetir o procedimento do *loop* e da atualização da matriz várias vezes. Vamos lá (vou tirar alguns comentários do código para que não fique tão grande):

```
import numpy as np

delta_x = 0.2       #delta x
delta_t = 0.01      #delta t

Temp = np.array([0.0,0.0,0.0,0.0,0.0,1.0])

print('Temperatura para k = 0:', Temp)

Temp_nova = np.copy(Temp)

for i in range(1,5):
    Temp_nova[i] = Temp[i] + (delta_t/(delta_x*delta_x))*(Temp[i+1]
                           - 2.0*Temp[i] + Temp[i-1])

Temp = np.copy(Temp_nova)

print('Temperatura para k = 1:', Temp)

for i in range(1,5):
    Temp_nova[i] = Temp[i] + (delta_t/(delta_x*delta_x))*(Temp[i+1]
                           - 2.0*Temp[i] + Temp[i-1])

Temp = np.copy(Temp_nova)

print('Temperatura para k = 2:', Temp)

for i in range(1,5):
    Temp_nova[i] = Temp[i] + (delta_t/(delta_x*delta_x))*(Temp[i+1]
                           - 2.0*Temp[i] + Temp[i-1])
```

```
Temp = np.copy(Temp_nova)

print('Temperatura para k = 3:', Temp)
```

```
Temperatura para k = 0: [0. 0. 0. 0. 0. 1.]
Temperatura para k = 1: [0. 0. 0. 0. 0.25 1. ]
Temperatura para k = 2: [0. 0. 0. 0.0625 0.375 1. ]
Temperatura para k = 3: [0. 0. 0.015625 0.125 0.453125 1. ]
```

Conseguimos avançar bem. Mas ainda é meio complicado. E se eu quisesse avançar 1000 passos de tempo? Eu teria que copiar essas linhas 1000 vezes. Vamos melhorar isso. Como é um bloco de repetição, vamos colocá-lo dentro de um *loop*. Temos, assim, um *loop* no espaço (o *loop* de dentro, da variável i) e um *loop* no tempo (o *loop* de fora, da variável k). Nosso código agora:

```
import numpy as np

delta_x = 0.2       #delta x
delta_t = 0.01      #delta t

Temp = np.array([0.0,0.0,0.0,0.0,0.0,1.0])

print('Temperatura para k = 0:', Temp)

Temp_nova = np.copy(Temp)

for k in range(1,4):
    for i in range(1,5):
        Temp_nova[i] = Temp[i] + (delta_t/(delta_x*delta_x))*(Temp[i+1]
                              - 2.0*Temp[i] + Temp[i-1])
    Temp = np.copy(Temp_nova)
    print('Temperatura para k =', k, ':', Temp)
```

```
Temperatura para k = 0: [0. 0. 0. 0. 0. 1.]
Temperatura para k = 1: [0. 0. 0. 0. 0.25 1. ]
Temperatura para k = 2: [0. 0. 0. 0.0625 0.375 1. ]
Temperatura para k = 3: [0. 0. 0.015625 0.125 0.453125 1. ]
```

Já está ficando bem melhor. E se eu quisesse resolver o problema usando 20 intervalos em vez de 5? E seu eu quisesse saber a resposta depois de 10 passos de tempo? Vamos ver como permitir essas modificações:

```
import numpy as np

N = 10     #Numero de intervalos no dominio
k_final = 10     #Numero de passos de tempo

L = 1.0     #Tamanho da barra

delta_x = L/N       #delta x
```

```
#Existe uma restricao para o delta_t, dado
#um delta_x. O delta_t tem que ser menor
#do que delta_x*delta_x/2. Por isso, vamos
#definir um delta_t que obedeca a essa
#restricao. No seu trabalho voce vai ver
#o que acontece quando essa restricao nao
#eh obedecida.

delta_t = 0.2*delta_x*delta_x      #delta t

#Vamos definir o vetor temperatura como sendo
#formado por zeros, com N+1 pontos.

Temp = np.zeros(N+1,float)

#No entanto, o ponto da extremidade direita, que
#eh o N, tem valor 1.

Temp[N] = 1.0

print('Temperatura em t = 0:', Temp)

Temp_nova = np.copy(Temp)

#Veja que agora os limites dos loops estao
#em funcao de k_final e N.

for k in range(1,k_final+1):
    for i in range(1,N):
        Temp_nova[i] = Temp[i] + (delta_t/(delta_x*delta_x))*(Temp[i+1]
                                - 2.0*Temp[i] + Temp[i-1])
    Temp = np.copy(Temp_nova)

#Vamos escrever apenas o resultado final.
print('Temperatura em t =', k*delta_t,':', Temp )
```

```
Temperatura em t = 0: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
Temperatura em t = 0.020000000000000004 : [0.00000000e+00 3.27680000e-06 4.88448000e-
05 4.50764800e-04 2.88839680e-03 1.36701952e-02 4.96746496e-02 1.42427546e-01 3.29289626e-
01 6.26181530e-01 1.00000000e+00]
```

Para visualizar o resultado o melhor mesmo é plotar o gráfico. Vamos precisar do vetor x para plotar o gráfico. Vamos lá (vou tirar os comentários feitos no código anterior):

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

N = 10
k_final = 10
```

```
L = 1.0

x = np.linspace(0.0,L,N+1) #Vetor x, usado para plotar

delta_x = L/N

delta_t = 0.2*delta_x*delta_x

Temp = np.zeros(N+1,float)

Temp[N] = 1.0

Temp_nova = np.copy(Temp)

for k in range(1,k_final+1):
    for i in range(1,N):
        Temp_nova[i] = Temp[i] + (delta_t/(delta_x*delta_x))*(Temp[i+1]
                                - 2.0*Temp[i] + Temp[i-1])
    Temp = np.copy(Temp_nova)

t = k*delta_t #Tempo atual, usado no titulo do grafico

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

fig.suptitle('t = %.3f' %t, fontsize=18, fontweight='bold')
ax.set_ylabel('$T$', fontsize=18)
ax.set_xlabel('$x$', fontsize=18)

plt.plot(x, Temp, '-r', lw = 4)

plt.savefig('figura.pdf', format='pdf', dpi=1200, bbox_inches='tight')

plt.show()
```

O RESULTADO É O GRÁFICO

O resultado é o gráfico da figura (15). Apenas como ilustração, o que eu vejo no Spyder está na figura (16).

Finalizando. Esse tutorial acaba aqui. Agora você vai trabalhar em cima desse código para melhorá-lo e para resolver os problemas propostos na próxima seção. Talvez seja interessante colocar o *loop* no tempo com um *while*, e assim poder definir o tempo final de parada. Também dá pra melhorar o gráfico de saída e você pode tentar fazer gráficos $3D$, incluindo o eixo do tempo. Fique à vontade para fazer todas as modificações/melhorias que achar necessário e para deixar o código com a sua cara.

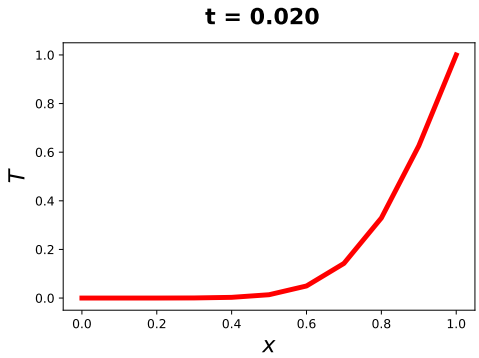

Figura 15: Temperatura em função de x para $t = 0,02$.

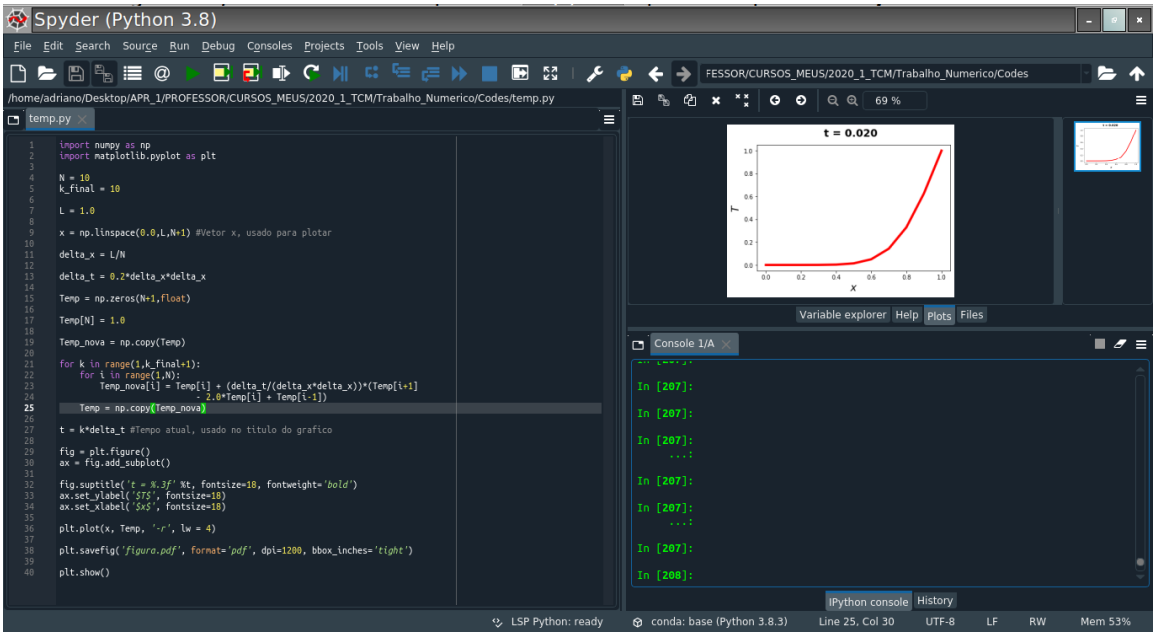

Figura 16: Spyder.

5 Estrutura do Trabalho e Data de Entrega

O objetivo aqui é resolver a equação do calor com diferentes configurações de condições iniciais e de contorno. Você vai resolver o mesmo problema que a gente desenvolveu aqui no roteiro, mas com condições de contorno e iniciais diferentes. Os problemas relacionados ao trabalho são apresentados abaixo. **O trabalho vale 5% extra na média final** e deve ser entregue na forma de um **artigo científico** (com **Título, Autor, Resumo, Introdução, Fundamentação Teórica, Metodologia Numérica, Resultados, Conclusões e Referências Bibliográficas**) juntamente com uma **apresentação gravada** do trabalho (de no máximo 15 minutos). Você vai fazer um vídeo apresentando o seu artigo.
Fique à vontade para usar a linguagem de programação que você quiser.
Os seguintes problemas deverão ser resolvidos e discutidos:

Problema 1:

$$\frac{\partial T(x,t)}{\partial t} = \frac{\partial^2 T(x,t)}{\partial x^2} \ , \qquad t > 0 \ , \qquad 0 < x < 1 \ . \tag{39}$$

$$T(0,t) = 0 \qquad \text{e} \qquad T(1,t) = 0, \qquad t \geq 0 \tag{40}$$

$$T(x,0) = 1, \qquad 0 < x < 1 \tag{41}$$

Solução exata:

$$T(x,t) = \sum_{n=1}^{\infty} \frac{4}{(2n-1)\pi} \sin\left[(2n-1)\pi x\right] \exp\left[-(2n-1)^2 \pi^2 t\right] \ . \tag{42}$$

Problema 2:

$$\frac{\partial T(x,t)}{\partial t} = \frac{\partial^2 T(x,t)}{\partial x^2} \ , \qquad t > 0 \ , \qquad 0 < x < 1 \ . \tag{43}$$

$$T(0,t) = 1 \qquad \text{e} \qquad T(1,t) = 0, \qquad t \geq 0 \tag{44}$$

$$T(x,0) = 0, \qquad 0 < x < 1 \tag{45}$$

Solução exata:

$$T(x,t) = 1 - x - \sum_{n=1}^{\infty} \frac{2}{n\pi} \sin\left[n\pi x\right] \exp\left[-n^2 \pi^2 t\right] \ . \tag{46}$$

Problema 3:

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} \ , \qquad t > 0 \ , \qquad 0 < x < 2 \ . \tag{47}$$

$$T(0,t) = T(2,t) = 0 \, , \quad t \geq 0 \tag{48}$$

$$T(x,0) = \sin\left(\frac{\pi}{2} x\right) \ , \qquad 0 < x < 2 \tag{49}$$

Solução exata:

$$T(x,t) = \exp\left[\frac{-\pi^2 t}{4}\right] \sin\left[\frac{\pi}{2} x\right] \ . \tag{50}$$

Apresente a solução de cada problema em diferentes instantes de tempo (escolha pelo menos 4 instantes de tempo para cada caso: um no começo, dois intermediários e um representando o regime permanente, ou seja, para t grande). Comente o que acontece com o perfil de temperatura quando $t \to \infty$. É o esperado? Compare a solução em cada tempo com a solução analítica. Resolva com diferentes valores de Δx e compare os resultados. Em todos os problemas deve ser analisada também a dependência da solução com o tamanho do Δx e do Δt escolhidos. **Atenção: devemos ter sempre $\Delta t < \Delta x^2/2$.** O que acontece quando essa condição não é obedecida? Faça alguns testes. Investigue também qual é a influência de α na solução. **Os códigos desenvolvidos devem ser enviados separadamente.**

A entrega do trabalho numérico deverá ser feita na última semana de aula. A data exata será definida ao longo do semestre.