Física Computacional - Prática 9

Questão 4

Alex Enrique Crispim

O método FTCS (*foward-time centered-space*) consiste numa forma de se transformar uma equação diferencial partial dependente de uma variável denotada tempo (t) e uma outra variável, a qual vamos chamar posição[1]

Seja

$$\frac{\partial T}{\partial t} = f(T(t,x),\, \partial^2 T/\partial x^2\ ,t,x) \tag{1}$$

a equação que desejamos resolver. Comecemos por truncar a série de Taylor em primeira ordem, de forma a trocar $\partial^2 T/\partial x^2$ por

$$\frac{\partial^2 T}{\partial x^2}(t,x) = \frac{T(t,x-a)+T(t,x+a)-2T(t,x)}{a^2}.$$

Para cada x fixo, temos

$$\frac{\partial T}{\partial t}(t,x) = f(T(t,x), T(t,x-a), T(t,x+a), t, x).$$

Da forma como temos a equação acima, para cada x, t varia de forma independete de x, de tal forma que podemos identificar a derivada parcial com relação ao tempo com a derivada total. Estamos, então, na presença de uma equação diferencial ordinária (EDO). Podemos fazer uso dos métodos já conhecidos para a solução de EDO's.

Como truncamos a série em primeira ordem, nosso erro no cálculo de $\partial^2 T/\partial x^2$ é de segunda ordem, de forma que utilizarmos um método com erro de terceira ordem ou mais (RK2 ou RK4, por exemplo), não nos dará melhores resultados que um método cujo erro é de primeira ordem (Euler, por exemplo, cujo truncamente ocorre em primeira ordem também).

A seguir apresentamos o resultado obtido para algumas temperaturas. O programa feito (disponível em `https://github.com/AlexEnrique/pratice-9-and-10`) fora feito de forma diferente do pedido pelo exercício. Explicaremos o motivo de tal escolha.

[1] A generalização para mais outra variáveis "espaciais"é feita trivialmente, trocando-se $\frac{\partial}{\partial x}$ por $\boldsymbol{\nabla}$.

Ao construir o programa, fizemos uso da técnica de *passagem de argumentos via linha de comando*. Basicamente, passamos argumento para o programa diretamente na hora de chamá-lo, escrevendo-os logo após o comando de execução do programa. Por exemplo, digitar `./programa.x 3.12 22 a` chama o programa de nome `programa.x`, passando os argumentos `3.12, 22` e `a` para o programa. Tais argumentos são lidos e podem ser utilizados pelo programa da forma como se desejar.

Para fazermos isso, definimos a função `main` da seguinte forma

```c
    int main (int argc, char *argv[]);
```

A variável `argc` é responsável por contar o número de parâmetros passados via linha de comando, somado a 1. Esse 1 a mais se refere a própria chamada do programa. Os parâmetros passados são armazenados como um tipo `char *` no array `argv[]`. O primeiro elemento deste array, `argv[0]` guarda a chamada do programa (para o exemplo anterior, `argv[0] == ./programa.x`). Os demais elementos são armazenados na ordem em que foram passados. Interessante citar a opção de se utilizar as funções `atoi()` e `atof()` (e outras) da biblioteca `<string.h>` para converter tipos `char *` em inteiros e pontos flutuantes (dupla precisão), o que nos dá maiores opções para a passagem de parâmetros via linha de comando.

Escolheu-se aceitar 3 parâmetros: o tempo final (denominado `tEnd`), uma opção de plotagem (`<plot option>`) e o nome do arquivo de (dados de) saída (`<file name>`). Requer-se que, pelo menos o parâmetro de tempo final seja passado. Com efeito, se não se passa tal parâmetro, a seguinte mensagem de erro é impressa na tela:

```
============================================================================
= ERROR: you have to pass the end time via command line                    =
= Call the programa and write the end time after the name                  =
=       (example: $ ./programname.x 3.1415)                                =
= The above example calls the program passing "3.1415" as the end time     =
=                                                                          =
= You can pass others parameters via command line. The posibilities are:   =
= the "file name" and a "plot option", as show below:                      =
=                                                                          =
=       $ ./programname.x <tEnd> <plot option> <file name>                 =
=                                                                          =
= If <plot option> is '1', the program will call the gnuplot, otherwise, no.  =
= Also, if the gnuplot will be call, the program creates a file called \   =
= "diffus.gnuscript" at the current folder.                                =
=                                                                          =
= <file name> could be any string representing the output file name or '-1'.  =
= If you pass '-1', the program will output the file as "diffus.t=<tEnd>"  =
= In the case of <file name> not beign passed, the program will output a file =
= called "diffus.dat".                                                     =
============================================================================
```

Optou-se por fazer uso de tal método na criação do programa de forma a fazer um programa de uso mais amplo e, em certo sentido, mais simples. Para se fazer o que se pede no enunciado do exercício, bastaria criar um outro código que chamasse o mesmo programa e passasse os valores requeridos de `tEnd`. Abaixo temos um exemplo (quase que no formato de um pseudo-código).

```
int main () {
  double t[] = [t1, t2, t3]; // colocar os tempos requeridos

  for (i = 0; i < t.length; i++) {
    snprintf(buffer, BUFFER_SIZE, "./program.exe %lf 0 -1", t[i]);
    system(buffer) // passa a string buffer para o terminal, como comando
  }
}

```

O reaproveitamento do programa se torna maior com o uso dos argumentos da função main().

Abaixo temos um gráfico para os três valores de tempo pedidos no enunciado.

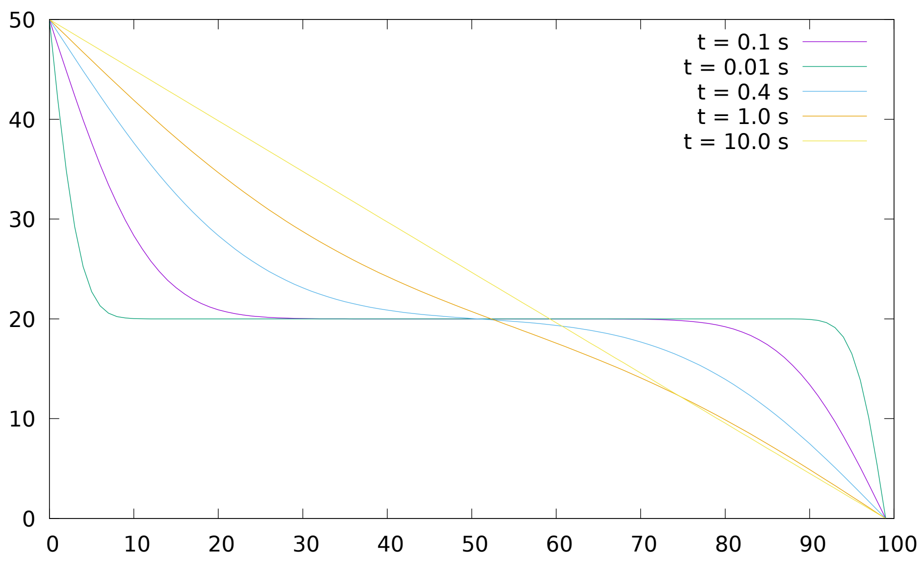

Figura 1: Solução para a equação de difussão