Projeto : Física Computacional

Miguel Madeira - 93402 Pedro Tavares - 93410
Grupo F04 - Sexta-feira - 2019/2020
Instituto Superior Técnico
Física Computacional
Prof. Rui Coelho

4 de Janeiro de 2020

Com o objetivo de se estudar um essemble de partículas carregadas sujeitas a um campo elétrico, provocado pelas mesmas, realizou-se uma simulação unidimensional em C++, em conjunção com as bibliotecas ROOT. Esta permite a visualização da dinâmica do sistema por vários meios, nomeadamente, espaço de fases, potencial e densidade de partículas, quer em tempo real, quer para um instante em específico. De forma a melhor compreender este relatório, é importante ler o ficheiro README, seguir o código dos métodos mencionados, observar as imagens submetidas e, porventura, executar o programa.

As equações normalizadas que regem o sistema físico em estudo são as seguintes :

$$\frac{d^2 r_i}{dt^2} = -E(r_i) \tag{1}$$

$$E(x) = -\frac{d\phi(x)}{dx} \tag{2}$$

$$\frac{d^2\phi(x)}{dx^2} = \frac{n(x)}{n_0} - 1 \tag{3}$$

Sendo que, para um dado instante de tempo, r_i representa a posição normalizada do eletrão-`i`, contido na *caixa periódica* de extensão `L = x_max - x_min`, e $E(x)$, $\phi(x)$ e $n(x)$ representam, respetivamente, o campo elétrico, o potencial e a densidade de eletrões normalizados, em função da posição $x \in$ [`x_min, x_max`].

Parte I. Condições Iniciais e Condições Fronteira

1. Condições Iniciais

A posição e a velocidade iniciais de cada partícula são obtidas através do método **FdistV()**. Para o cálculo das posições, gera-se uma distribuição uniforme no intervalo [`x_max - x_min`], recorrendo-se a uma distribuição uniforme em $[0, 1]$. Quanto às velocidades, seguiu-se a Transformada de Box-Muller [1], que permite gerar distribuições normais independentes, a partir de uma distribuição uniforme, aplicando-se, neste caso, a $e^{-\frac{(v-v_b)^2}{2}}$ e $e^{-\frac{(v+v_b)^2}{2}}$. Para que a concretização do cálculo associado a cada uma das distribuições normais tenha a mesma probabilidade, gera-se um número aleatório entre 0 e 1 e verifica-se se este é inferior (ou superior) a 0.5.

O método **FdistV()** apresenta, ainda, as opções `save_plot` e `show_plot` (argumentos do método) que permitem, respetivamente, guardar um *plot* de um histograma normalizado com

as velocidades geradas, juntamente com a função analítica de distribuição pretendida, ou, por sua vez, mostrá-los numa janela (`TApplication`) (cf. **Parte III**).

O *plot* da função de distribuição analítica não se encontra representado no mesmo sistema de eixos que o histograma, uma vez que, deste modo, é mais fácil discernir o histograma de velocidades do gráfico da função analítica. De forma a facilitar a comparação entre os dois, ambos foram normalizados através do produto pelo inverso do respetivo integral.

2. Condições Fronteira

Com o objetivo de garantir que os eletrões estão confinados ao domínio pretendido (`[x_min, x_max]`), foi imposta uma condição fronteira à posição dos mesmos. No método **TimeStep()**, depois da atribuição das posições e velocidades de cada uma das partículas, aplicam-se as condições fronteira através das condições:

Listing 1: *Condições Fronteira: Posição*

```
1    if (x_vpart[i][0] > x_max) x_vpart[i][0] = x_vpart[i][0] - L;
2    if (x_vpart[i][0] < x_min) x_vpart[i][0] = x_vpart[i][0] + L;
```

Ou seja, atingido um dos limites da *caixa*, a partícula é deslocada para o extremo oposto da mesma, sendo que a sua velocidade instantânea não é alterada.

Desta forma, atendendo a esta periodicidade do movimento e ao enunciado do projeto, é necessário, também, garantir que:

$$n(\texttt{x\_min}) = n(\texttt{x\_max}) \qquad E(\texttt{x\_min}) = E(\texttt{x\_max}) \qquad \phi(\texttt{x\_min}) = \phi(\texttt{x\_max}) = 0$$

Parte II. Métodos Numéricos e Algoritmos

1. Densidade de Eletrões - UpdateDensGrid()

De modo a determinar-se a densidade de eletrões no domínio espacial em causa, discretizou-se o mesmo numa grelha uniforme composta por `ngrid` pontos, distanciados entre si por `hgrid` unidades de comprimento. Assim, a densidade em cada ponto `xgrid[j]` da grelha, com `j` $= 0, \ldots,$ `ngrid` $- 1$, foi definida em função da posição relativa de cada partícula a estes pontos, tal como descrito no enunciado do projeto. Para tal, definiu-se o método **UpdateDensGrid()** com o intuito de se fixar os valores de `dens_grid[j]`, em cada ponto da grelha, num dado instante de tempo.

Desta forma, com o objetivo de se determinar o intervalo da grelha espacial tal que `xgrid[j]` $<$ `x_vpart[i][0]` $<$ `xgrid[j + 1]`, para cada posição `x_vpart[i][0]` da partícula-`i`, implementou-se, também, o método **BinarySearch()**, que permite uma procura mais rápida deste intervalo, quando comparado com o método linear. Este método auxiliar é muito semelhante ao método da bisseção mas, agora, aplicado a um caso discreto.

Desta forma, determinado este intervalo para uma dada partícula, a atribuição dos valores de `dens_grid[j]` e `dens_grid[j+1]` foi efetuada de acordo com as expressões:

$$\texttt{dens\_grid[j]} \mapsto \texttt{dens\_grid[j]} + \frac{\texttt{xgrid[j]} - \texttt{x\_vpart[i][0]}}{\texttt{hgrid}^2}$$

$$\texttt{dens\_grid[j+1]} \mapsto \texttt{dens\_grid[j+1]} + \frac{\texttt{x\_vpart[i][0]} - \texttt{xgrid[i+1]}}{\texttt{hgrid}^2}$$

No entanto, esta estratégia assume que a densidade nos extremos da *caixa* não são necessariamente iguais, não estando, por isso, de acordo com a periodicidade do sistema. Assim, é

necessário proceder-se à seguinte correção:

$$\texttt{dens\_grid[0]} = \texttt{dens\_grid[ngrid - 1]} \mapsto \texttt{dens\_grid[ngrid - 1]} + \texttt{dens\_grid[0]}$$

Por fim, efetuou-se a devida normalização da densidade, congruente com a **Equação 3**:

$$\texttt{dens\_grid[j]} \mapsto \frac{\texttt{dens\_grid[j]}}{\texttt{dens\_n0}} - 1 \qquad \text{com} \qquad \texttt{dens\_n0} = \frac{\texttt{npart}}{\texttt{x\_max} - \texttt{x\_min}}$$

2. Potencial - UpdatePotGrid()

Uma vez estabelecida a densidade em cada ponto da grelha espacial (`dens_grid`), e atendendo às condições fronteira para o potencial, é necessário resolver a **Equação 3**, que permite determinar o potencial (`pot_grid`) nestes mesmos pontos (`xgrid`).

A partir do método de diferenças finitas para BVPs, tem-se que:

$$\frac{d^2\phi(x_j)}{dx^2} \approx \frac{\phi(x_j + h) - 2\phi(x_j) + \phi(x_j - h)}{h^2} = \frac{n(x_j)}{n_0} - 1 \qquad (j = 1,\ \ldots,\ N-1)$$

Em que se assume uma partição do intervalo $[\texttt{x\_min},\ \texttt{x\_max}]$ de N subintervalos iguais de comprimento h, de tal forma que : $x_j = \texttt{x\_min} + jh$, com $j = 0,\ \ldots,\ N-1$.

Neste caso, fazendo-se $N = \texttt{ngrid} - 1$ e $h = \texttt{hgrid}$, facilmente se obtém que:

$$\texttt{pot\_grid[j-1]} - 2 \cdot \texttt{pot\_grid[j]} + \texttt{pot\_grid[j+1]} = \texttt{hgrid}^2 \cdot \texttt{dens\_grid[j]} \qquad (\texttt{j} = 1,\ \ldots,\ \texttt{ngrid - 2})$$

Deste modo, considerando as condições fronteira $\texttt{pot\_grid[0]} = \texttt{pot\_grid[ngrid - 1]} = 0$, tem-se, finalmente, em notação matricial, que:

$$\begin{pmatrix} -2 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ \vdots & \cdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \cdots & 0 & 1 & -2 & 1 \\ 0 & \cdots & 0 & 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} \texttt{pot\_grid[1]} \\ \texttt{pot\_grid[2]} \\ \texttt{pot\_grid[3]} \\ \vdots \\ \texttt{pot\_grid[ngrid - 3]} \\ \texttt{pot\_grid[ngrid - 2]} \end{pmatrix} = \texttt{hgrid}^2 \begin{pmatrix} \texttt{dens\_grid[1]} \\ \texttt{dens\_grid[2]} \\ \texttt{dens\_grid[3]} \\ \vdots \\ \texttt{dens\_grid[ngrid - 3]} \\ \texttt{dens\_grid[ngrid - 2]} \end{pmatrix}$$

Com isto, recorrendo ao método **TridiagonalSolver()** da classe **EqSolver**, desenvolvida ao longo do semestre, foi possível determinar `pot_grid`, i.e., o potencial em cada ponto de `xgrid`.

3. Campo Elétrico - Spline3Interpolator::Deriv()

A partir da **Equação 1** e **Equação 2**, é óbvio que:

$$\frac{d^2 r_i}{dt^2} = \frac{d\phi(r_i)}{dx} \tag{4}$$

Ou seja, as sucessivas posições de cada partícula-`i`, ao longo do tempo de simulação, são determinadas pelo valor da derivada do potencial, avaliada na posição (`x_vpart[i][0]`) que cada uma ocupa, nesse instante de tempo.

Uma vez que o método **UpdatePotGrid()** apenas permite determinar o potencial nos pontos `xgrid` da grelha espacial, é necessário recorrer-se a métodos de interpolação dos pontos $\{(\texttt{xgrid[j]},\ \texttt{pot\_grid[j]})\}$, de modo a ser possível avaliar-se o potencial (e, consequentemente, a sua derivada), em qualquer ponto do intervalo $[\texttt{x\_min},\ \texttt{x\_max}]$.

Deste modo, recorreu-se a *spline cubic interpolation*, utilizando-se, para este efeito, a classe **Spline3Interpolator**. Como é sabido, este método requer a introdução de duas constantes (α e β) tais que:

$$\phi''(\texttt{x\_min}) = \alpha \qquad \text{e} \qquad \phi''(\texttt{x\_max}) = \beta$$

Pelo que, de acordo com a **Equação 3** :

$$\alpha = \texttt{dens\_grid[0]} \qquad \text{e} \qquad \beta = \texttt{dens\_grid[ngrid - 1]} \qquad (\implies \alpha = \beta)$$

Portanto, foi necessário adicionar o método **Deriv()** à classe **Spline3Interpolator**, dado que a mesma não possuía um método capaz de determinar a derivada nos pontos do domínio de interpolação [`x_min`, `x_max`]. A implementação deste método é muito semelhante à realizada, durante o semestre, para o método **Interpolate()**, diferindo apenas na expressão que permite obter, neste caso, a derivada da função interpoladora, que é dada, obviamente, pela derivada analítica das funções de *spline*, subjacentes a este método de interpolação.

4. Posição / Velocidade - TimeStep()

A **Equação 1** pode ser escrita como um sistema de duas ODEs, de tal modo que:

$$\texttt{vector<TFormula> ODE} = \begin{cases} \frac{dr_i}{dt} = v_i \\ \frac{dv_i}{dt} = \frac{d\phi(r_i)}{dx} \end{cases}$$

Onde $r_i = \texttt{x\_vpart[i][0]}$ representa, tal como referido anteriormente, a posição e $v_i = \texttt{x\_vpart[i][1]}$ a velocidade instantânea da partícula-`i`. Para evoluir no tempo este sistema de ODEs, foi utilizada a classe **ODEsolver**, recorrendo-se ao método de Runge-Kutta 4, implementado no método **RK4_iterator()** desta classe.

O membro privado `ODEsolver Equation`, da classe **PIC**, tem acesso a um `vector<TFormula> ODE` (cf. *constructor* da classe **PIC**), cujo parâmetro `[0]` de `ODE[1]` é, no método **TimeStep()**, sucessivamente atualizado em função da derivada do potencial ($\propto E(r_i)$) na posição `x_vpart[i][0]`. Assim, para cada partícula-`i`, faz-se uso do método **UpdateParameter()**, da classe **ODEsolver**, para atualizar o valor de `[0]` da `TFormula ODE[1]` e, de seguida, a partir de **RK4_iterator()**, avança-se um passo no tempo.

Após este avanço no tempo, é efetuado o devido ajuste das posições das partículas, de modo a garantir a concordância destas com a amplitude e periodicidade do domínio espacial, tal como descrito na secção referente às condições fronteira do sistema.

Parte III. Criação e Gravação de Figuras

Os métodos públicos **Poisson()**, **Density()** e **PhaseSpace()** implementados na classe **PIC**, quando manipulados corretamente, permitem o *plot* em tempo real da simulação, a gravação de imagens em qualquer instante de tempo da simulação, tendo em conta, evidentemente, o passo no tempo considerado em **TimeStep()**, e o *plot* de um gráfico *estático*, durante o decorrer da simulação.

De forma a preparar as `TCanvas` e `TGraphs` para o desenho das figuras, foram criados os métodos privados: **SetAppCanvas()** ; **SetGraphs()** ; **SetSaveCanvas()** ; **SetSaveDist()**. Estes métodos são muito semelhantes entre si e têm um fim, principalmente, cosmético. Nestes é definido o tamanho das `TCanvas` e a divisão das mesmas em `TPads`, sendo que, para os `TGraphs`, são definidos títulos, limites dos eixos, cor, estilo e tamanho dos *markers*.

A declaração dos *Data Members* `Save_dist, Save_c, App e App_c`, como membros da classe **PIC**, tem como objetivo a reutilização das mesmas `TCanvas` nas várias iterações da simulação e, se for esse o caso, por mais do que um objeto. Note-se que `Save_dist` e `Save_c` estão

declaradas antes da `TApplication App`, sendo necessário que assim se mantenha. Caso uma `TApplication` estiver ativa quando qualquer `TCanvas`, declarada posteriormente, é alterada, abre-se um *widget*, pelo que, quando se grava várias imagens e/ou se está a tentar observar outros *plots*, por exemplo, em tempo real, torna-se pouco prático e, muitas vezes, conflituoso.

Os métodos **Poisson()**, **Density()**, **Plot_Phase_Space()** e **FdistV()** recebem como argumentos as variáveis *booleanas* `save_plot` e `show_plot`. Quando `save_plot = true`, é gravada uma imagem (com extensão `.eps`) dessa iteração, com o nome apropriado, associado ao instante de tempo correspondente. Por outro lado, quando `show_plot = true`, a `TCanvas App_c` é atualizada e é apresentado o gráfico da iteração que lhe corresponde. Não obstante, ambas as opções podem estar ativas em simultâneo. Com isto, caso se pretenda que o *plot* seja em tempo real, basta manter `bool show_plot = true`. Por conseguinte, caso se pretenda que o gráfico não surja no ecrã, esta deve permanecer a `false`. No entanto, se se pretender que o gráfico não evolua no tempo, permanencendo parado, deve-se definir a variável como `true` na iteração que se pretende observar e como `false` nas iterações seguintes. Pode-se, ainda, intercalar entre um *plot* em tempo real e uma imagem parada através de condições `if()`.

Parte IV. Resultados Obtidos

1. Caso I

No espaço de fases são visíveis, no início da simulação, 2 bandas centradas em cada uma das velocidades v_b, sendo que, no caso em estudo, estas são simétricas. À medida que a simulação avança no tempo, estas bandas deformam-se, até que, eventualmente, formam um vórtice. O movimento dos eletrões torna-se sucessivamente mais instável, deixando de ser possível distinguir as duas bandas iniciais. A formação de um vórtice no espaço de fases indica que, a partir de um determinado instante de tempo, os eletrões adquirem um movimento oscilatório provocado por disparidades na densidade de eletrões. O comprimento de onda destas oscilações corresponde ao diâmetro horizontal do vórtice.

Por outro lado, o potencial evolui quase de forma oposta. Inicialmente, apresenta uma forma irregular, mas, à medida que o tempo avança, esta vai ficando cada vez mais regular, adquirindo um comportamento, aproximadamente, *sinusoidal*. Note-se que a posição onde ocorre o máximo do potencial coincide com a posição do centro do vórtice, sendo que, em contrapartida, os *extremos* do vórtice surgem em pontos de mínimo do potencial. A concavidade do gráfico do potencial evolui de acordo com a função de densidade normalizada de eletrões.

2. Caso II

Com a diminuição do valor de `x_max`, a dinâmica temporal obtida é consideravelmente mais estável, pois verifica-se que a formação do vórtice ocorre cada vez mais tarde e o seu diâmetro horizontal vai, gradualmente, aumentando. Eventualmente, com `x_max` ≈ 29.2 ocorre a transição para um regime estável. Este valor pode variar um pouco dado que as condições iniciais têm, de facto, uma componente aleatória, mas, com `x_max` < 29.0, nunca se observou a formação de vórtices, mantendo-se as 2 bandas de partículas sempre bem definidas. Neste último caso, as partículas apresentam apenas movimento horizontal, tendo as bandas direção oposta: uma banda move-se para a *esquerda* e a outra para a *direita*, como seria de esperar.

Se, por outro lado, se aumentar o valor de `x_max`, a dinâmica é mais instável. Formam-se mais vórtices com o aumento do comprimento da *caixa* e a sua formação é mais rápida. Curiosamente, esses vórtices começam, de certa forma, a *unir-se*, formando um vórtice maior, sendo, no entanto, ainda possível distinguir os vórtices originais.

Referências

[1] Wikipedia Contributors (2019). Box–Muller transform. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform?fbclid=IwAR0W-SV1_QDQQ1snJy7l-CcChGoUQtrplmgLXEUEworvJkHBJyEk2FHL1XY [Accessed 26 Dec. 2019].

[2] GeeksforGeeks. Binary Search - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/binary-search/ [Accessed 25 Dec. 2019].

[3] Farside.ph.utexas.edu. Particle-in-cell codes. [online] Available at: http://farside.ph.utexas.edu/teaching/329/lectures/node96.html [Accessed 4 Jan. 2020].

[4] Root.cern. ROOT Reference Documentation. [online] Available at: https://root.cern/doc/master/index.html [Accessed 2019/2020].