TÉCNICO LISBOA

Simulação *Particle in Cell*

Alexandre Barbosa[1], Miguel Roldão[2]

Física Computacional

Mestrado Integrado em Engenharia Física Tecnológica

[1] alexandre.barbosa@tecnico.ulisboa.pt, 93362 [2] miguel.roldao@tecnico.ulisboa.pt, 93405

Introdução Teórica

A equação do movimento de um eletrão é:

$$\frac{d^2 r_i}{dt^2} = -\frac{qE(r_i)}{m_e} \qquad (1)$$

A partir da Lei de Gauss, obtemos:

$$\begin{cases} \nabla \cdot E = \frac{\rho}{\varepsilon_o} = \frac{e}{\varepsilon_o}(n_{ions} - n_{eletrons}) \\ E(x) = -\frac{d\phi(x)}{dt} \end{cases} \qquad (2)$$

E	campo elétrico
ρ	densidade de carga total
ε_o	permissividade elétrica do vácuo
n	densidade de partículas
ϕ	potencial elétrico

Normalização

$$t \longrightarrow t\,\omega_p^{-1} = t\,\sqrt{\frac{\varepsilon_0\, m_e}{n_0\, e^2}} \quad \text{e} \quad x \longrightarrow x\,\lambda_D \qquad (3)$$

$$n_0 = \frac{N}{L} \quad \text{onde} \quad L = x_{max} - x_{min} \qquad (4)$$

Discretização do Problema

Se a posição r_i do eletrão se encontrar no intervalo $[x_j, x_{j+1}]$, então a densidade nas células j e $j+1$ aumenta por:

$$\begin{cases} n_j \to n_j + \dfrac{x_{j+1} - r_i}{h^2} \\ n_{j+1} \to n_{j+1} + \dfrac{r_i - x_j}{h^2} \end{cases} \tag{5}$$

Implementação da Classe PIC

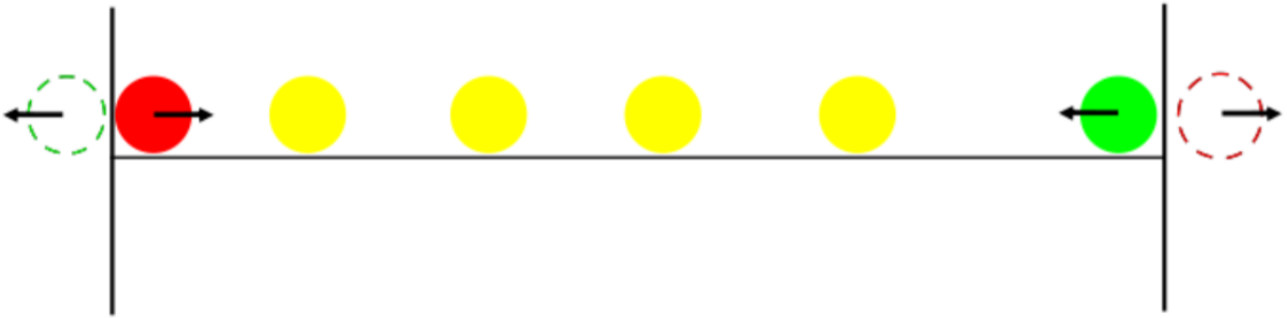
```
class PIC {
  public:
    PIC(std::vector<double> velocity, int Npart = 1000, double xmin = 0.0, double xmax = 1.0, int ngrid = 100);
    ~PIC(); // destructor
    // distribuição inicial de velocidades
    void FdistV(vector<double> veloc, bool save_plot = true, bool show_plot = false);
    // espaço de fases
    void Plot_Phase_Space(bool save_plot = true, bool show_plot = false);
    // cálculo do perfil de densidade na grelha espacial
    void Density(bool save_plot = true, bool show_plot = false);
    // cálculo do potencial na grelha espacial
    void Poisson(bool save_plot = true, bool show_plot = false);
    // avanço no tempo de todas as partículas
    void TimeStep(double dt);
    // discretiza o domínio espacial
    void SetGrid();
    // impõe as condições fronteira
    void CheckBoundaries();
    // retorna o tempo atual de simulação (unidades normalizadas)
    double GetTime() {return t;}
    PIC & operator= (const PIC &rhs); // assignement operator
```

Data Members da Classe

```
private:
  std::vector<double> velocity;  // vector com a velocidade dos 2 beams (positivo e negativo)
  int Npart;                     // = 1000: número total de partículas
  double xmin;                   // = 0.0 início do domínio espacial que contém as partículas
  double xmax;                   // = 1.0: fim do domínio espacial que contém as partículas
  int ngrid;                     // = 100: numero de pontos da grelha espacial onde calcular a densidade e potencial.
  double n0;                     // densidade média
  double * r;                    // array com a posição dos Npart eletrões
  double * v;                    // array com a velocidade dos Npart eletrões
  double* xgrid;                 // coordenadas da grelha (de xmin a xmax)
  double* density;               // densidade de partículas na grelha espacial
  double* potential;             // potencial elétrico a grelha espacial
  double* Efield;                // campo elétrico para as N partículas
  void Eletric_Effect(double dt);// calcula o campo elétrico, as novas velocidades e posições
  std::vector<TFormula> TF;      // vetor de TFormulas usadas nas equações diferenciais
  std::vector<ODEpoint> sol;     // vetor-solução da ODE que permite calcular a posicao e velocidade
  double t;                      // tempo atual de simulação
  static bool HasDrawnAnyPlots;  // para decidir se cria uma TApplication
```

Condições Fronteira

- Se $x_i > x_{max}$, então $x_i = x_i - L$.
- Se $x_i < x_{min}$, então $x_i = x_i + L$

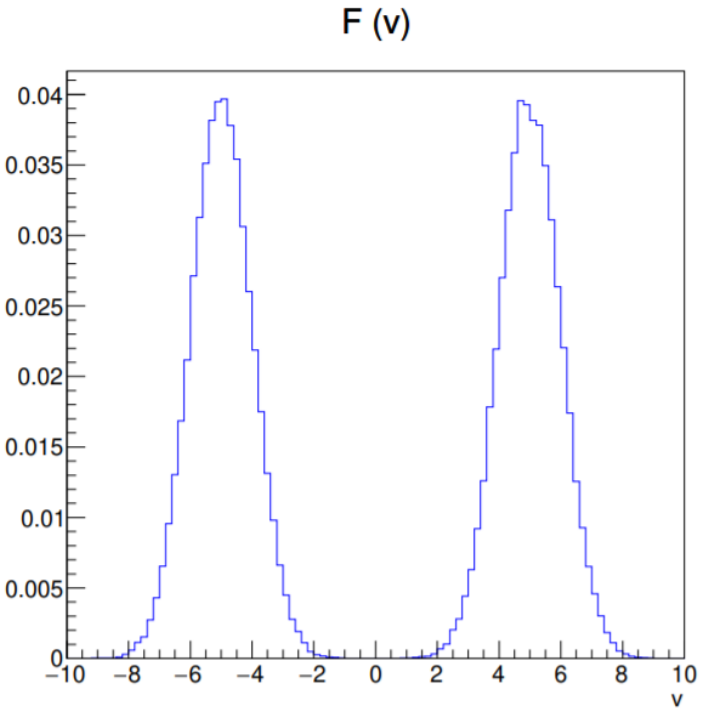


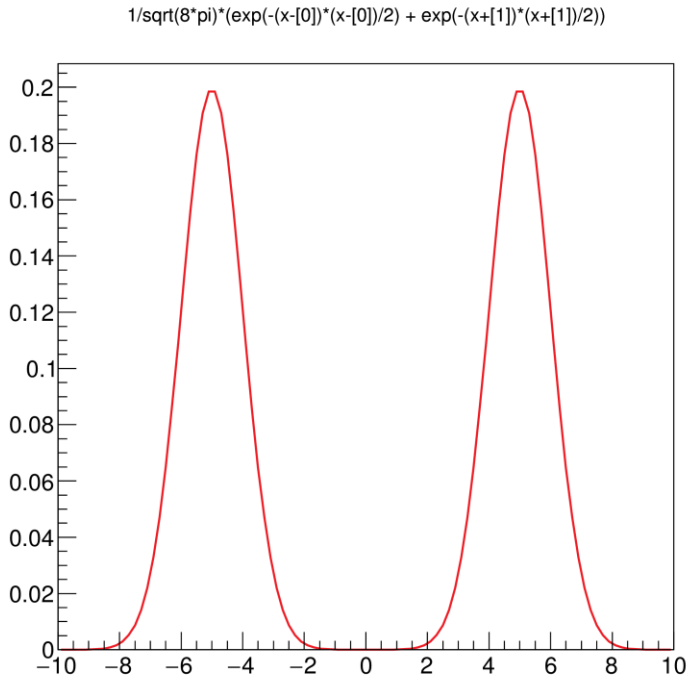
$$\phi(x_{min}) = \phi(x_{max}) = 0$$

Método de Aceitação-Rejeição

```
bool VelocityParameter;

// Método de Aceitação-Rejeição
for (int j = 0; j < Npart; j++)
{
  if (rgen.Uniform(0,1) > 0.5)
    VelocityParameter = true;

  else
    VelocityParameter = false;

  // Método de Box-Muller
  v[j] = sqrt(-2*log(rgen.Uniform(0,1)))*cos(2*PI*rgen.Uniform(0,1)) + veloc[VelocityParameter];
  histogram->Fill(v[j]);
}
```

Distribuição de Velocidades





Métodos Numéricos: Diferenças Finitas

A segunda derivada do campo elétrico ϕ pode ser escrita como:

$$\frac{d^2\phi}{dt^2} = \frac{\phi(x_i + h) - 2\phi(x_i) + \phi(x_i - h)}{h^2} \tag{6}$$

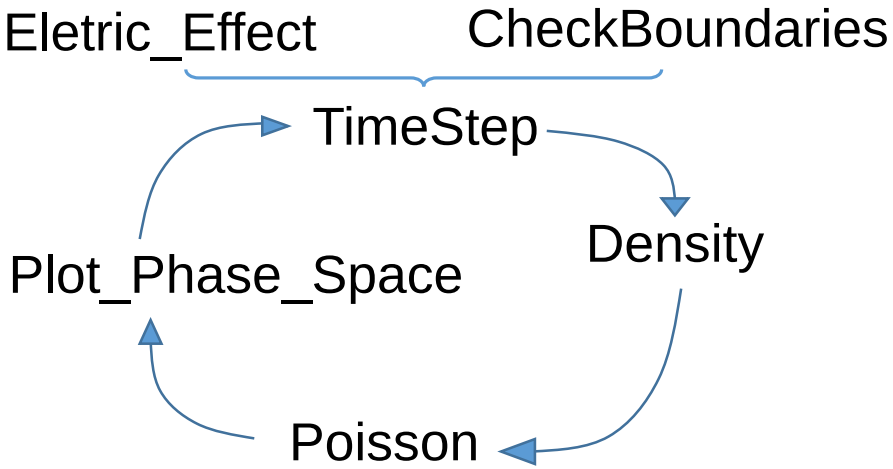
O cálculo do potencial resume-se a resolver a equação:

$$\frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & & \vdots \\ 0 & 1 & -2 & \ddots & \\ \vdots & & \ddots & \ddots & 1 \\ 0 & \cdots & & 1 & -2 \end{bmatrix} \vec{\phi} = \begin{bmatrix} 0 \\ n_1 - 1 \\ \vdots \\ n_{ngrid-2} - 1 \\ 0 \end{bmatrix} \tag{7}$$

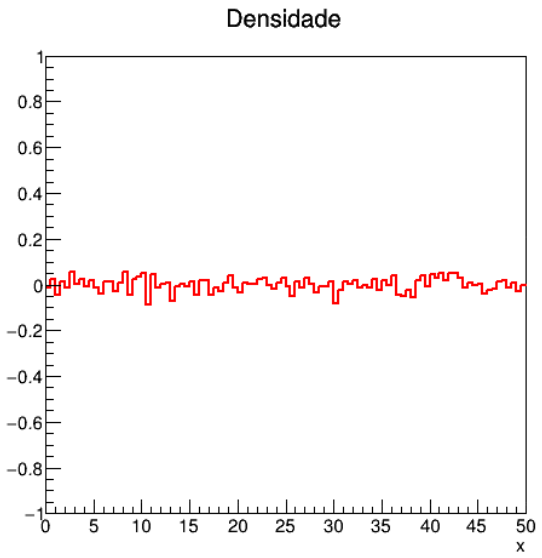
Métodos Numéricos: Runge-Kutta 4

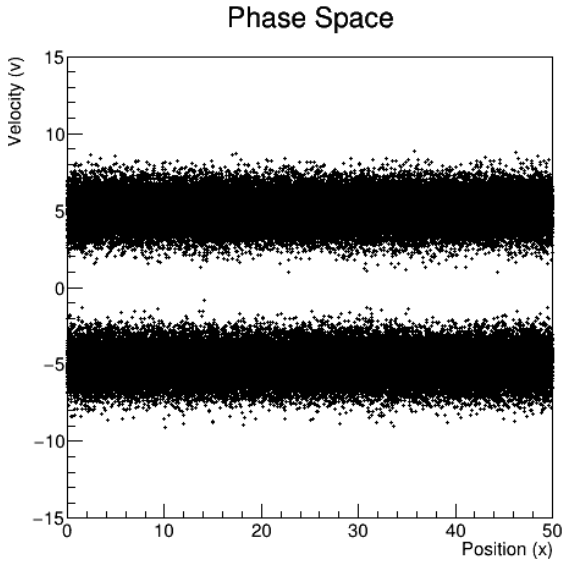
$$\begin{cases} \dfrac{dr_i}{dt} = v_i \\ \dfrac{dv_i}{dt} = -E(r_i) \end{cases} \tag{8}$$

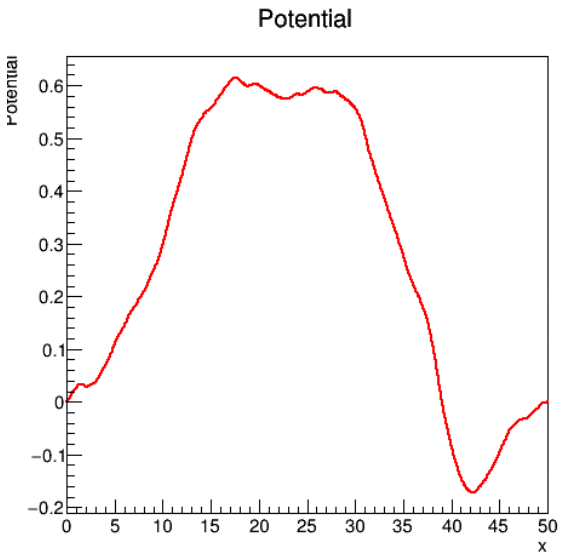
Avanço Temporal



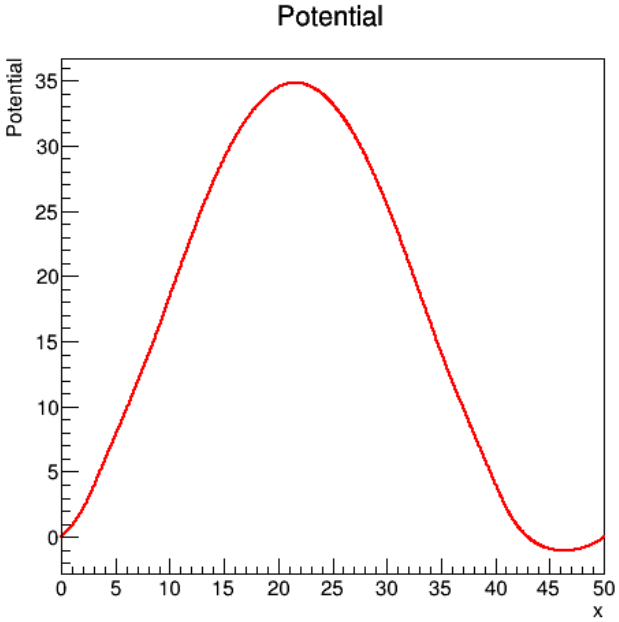
Simulação da Dinâmica de N = 10^5 Partículas



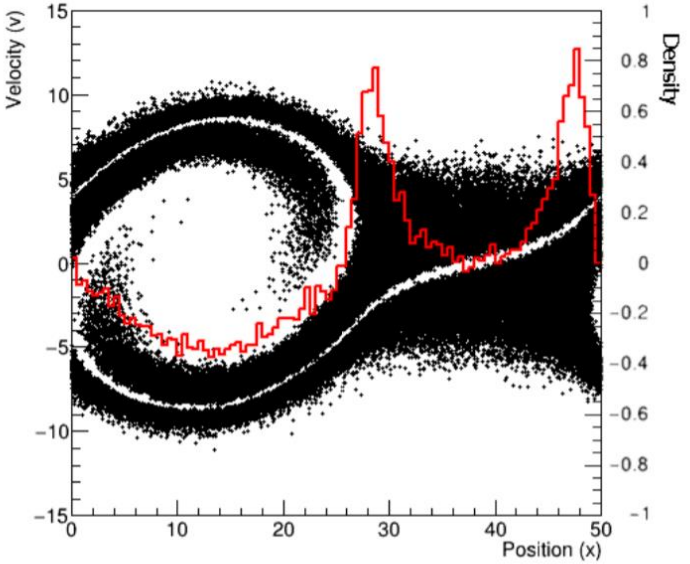




Caso I: L = 50.0 λ_D

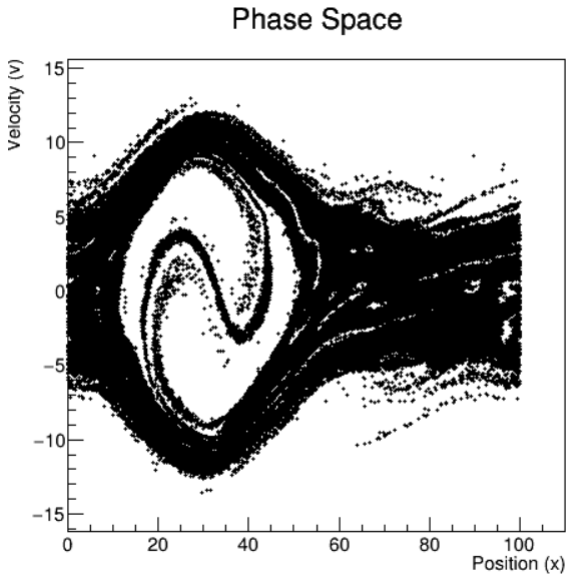


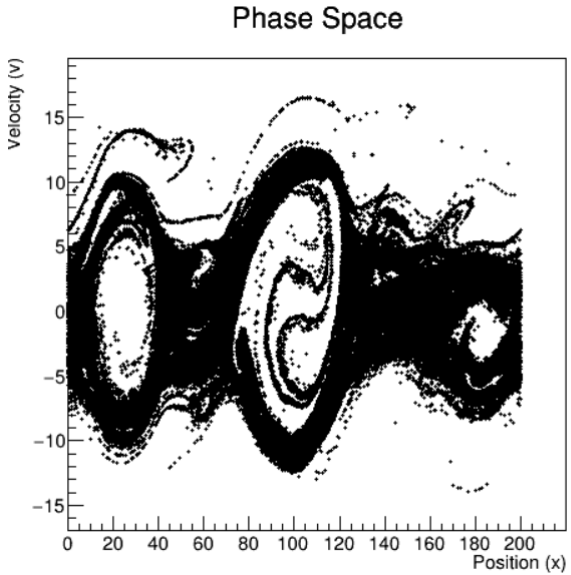
Potencial em t = 20.0 ω_p^{-1}



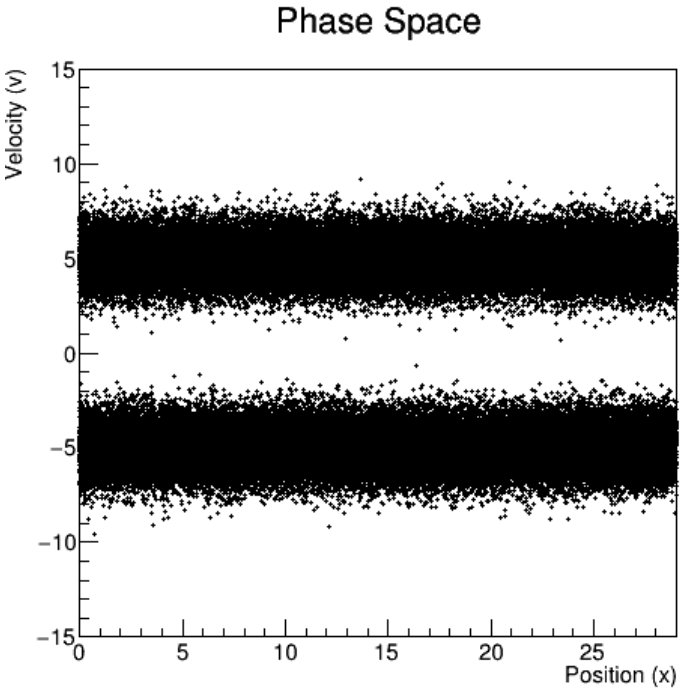
Densidade e Espaço de Fases em $t = 20\ \omega_p^{-1}$

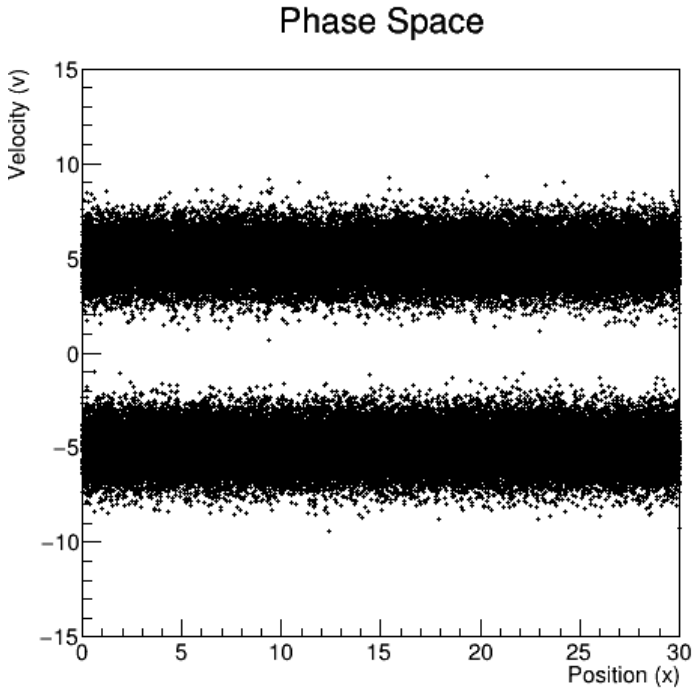
Caso II: Estabilidade (L > 50.0 λ_D)



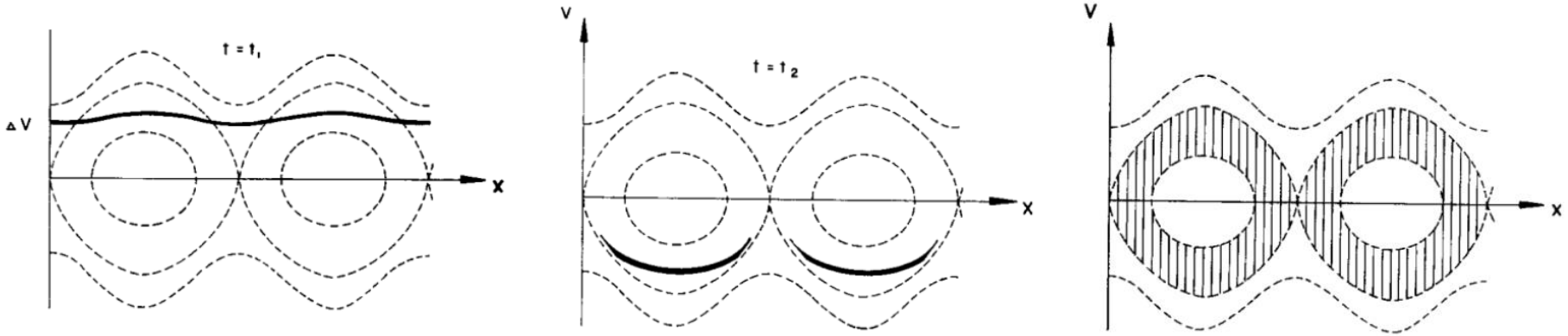


Caso II: Estabilidade ($L < 50.0\ \lambda_D$)





Instabilidade



W. E. Drummond, J. H. Malmberg, M. O'Neil e J. R. Thompson, "Nonlinear Development of the Beam-Plasma Instability". Em: The Physics of Fluids, **2242**. 13 (1970).

Limitações

$$60 \neq \infty$$

Limitações

100000000000000000000000

simulação **realidade**