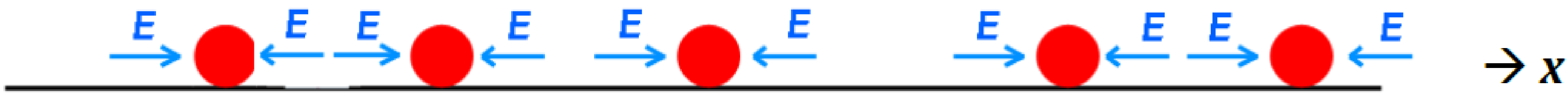
PIC
PARTICLE-IN-CELL

MIGUEL MADEIRA - 93402

PEDRO TAVARES - 93410

Prof . : Rui Coelho

SIMULAÇÃO : OBJETIVO



Equações da Dinâmica do Sistema

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

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

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

CLASSE PIC

```
class PIC {
  public :
    PIC () {;}; // default constructor
    PIC (vector<double> velocity, int Npart = 1000, double xmin = 0.0,
double xmax = 1.0, int Ngrid = 100); // constructor
    ~PIC(); // destructor

    void FdistV (vector<double> veloc, bool save_plot, bool show_plot = false);
    void Plot_Phase_Space (bool save_plot, bool show_plot = false);
    void Density (bool save_plot, bool show_plot = false);
    void Poisson (bool save_plot, bool show_plot = false);
    void TimeStep (double dt);

  protected :
    static int nshow;

    static TCanvas Save_dist;
    static TCanvas Save_c;
    static TApplication App;
    static TCanvas App_c;
```

CLASSE PIC

```cpp
private :
  vector<double> vel_b;
  double x_min;
  double x_max;

  int npart;

  int ngrid;
  double hgrid;

  double * xgrid;
  double * dens_grid;
  double * pot_grid;

  vector<ODEpoint> x_vpart;
  ODEsolver Equation;
```

```cpp
    FCmatrixBanded Tri_mat;
    double dens_n0;
    double L;

    void SetAppCanvas();
    void SetGraphs();
    void SetSaveCanvas();
    void SetSaveDist();

    void UpdateDensGrid();
    void UpdatePotGrid();

    friend int BinarySearch (int, double *, double);
};
```

CONDIÇÕES INICIAIS

FdistV()

Posição

Distribuição uniforme, em

$[\mathtt{x\_min},\ \mathtt{x\_max}]$

Velocidade

Distribuição $F(v)$

$$F(v) \propto e^{\frac{-(v-v_b)^2}{2}} + e^{\frac{-(v+v_b)^2}{2}}$$

```
for (int i = 0; i < npart; i++){
    aux[0] = x_min + (x_max - x_min) * rgen.Uniform(0,1);

    if (rgen.Uniform(0,1) < 0.5)
      aux[1] = sqrt(-2*log(rgen.Uniform(0,1)))*cos(2*M_PI*rgen.Uniform(0,1)) + veloc[0];
    else
      aux[1] = sqrt(-2*log(rgen.Uniform(0,1)))*cos(2*M_PI*rgen.Uniform(0,1)) - veloc[1];

    x_vpart.push_back(ODEpoint(0, aux, 2));
  }
```

Transformada de Box-Muller

CONDIÇÕES FRONTEIRA

Posição

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

Densidade

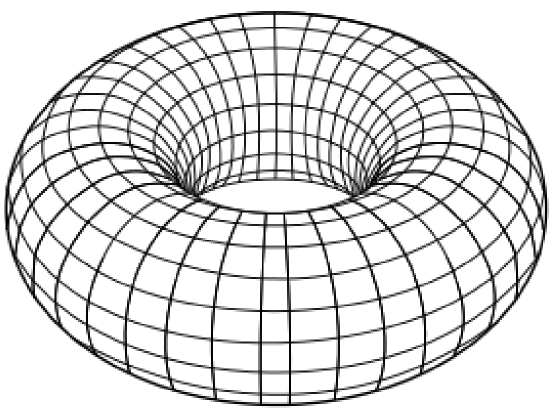
$$n(\texttt{x\_min}) = n(\texttt{x\_max})$$

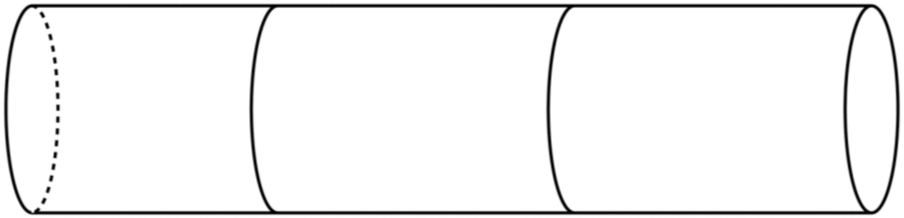
Campo Elétrico

$$E(\texttt{x\_min}) = E(\texttt{x\_max})$$

Potencial

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





DENSIDADE

UpdateDensGrid()

BinarySearch()

Método da Bisseção Discreto

$$n_j \mapsto n_j + \frac{x_{j+1} - r_i}{h^2}$$

$$n_{j+1} \mapsto n_{j+1} + \frac{r_i - x_j}{h^2}$$

Condições Fronteira

$$n_{j=0} = n_{\texttt{ngrid - 1}} \mapsto n_{j=0} + n_{\texttt{ngrid - 1}}$$

$$n_j \mapsto \frac{n_j}{n_0} - 1$$

$$n_0 = \frac{\texttt{npart}}{\texttt{x\_max} - \texttt{x\_min}}$$

```
int j = 0;

for (int i = 0; i < ngrid; i++){
  dens_grid[i] = 0;
} // reset da densidade de partículas

for (int i = 0; i < npart; i++){
  j = BinarySearch(ngrid, xgrid, x_vpart[i][0]);

  dens_grid[j] += (xgrid[j + 1] - x_vpart[i][0]) / (hgrid * hgrid);
  dens_grid[j + 1] += (x_vpart[i][0] - xgrid[j]) / (hgrid * hgrid);
}
```

POTENCIAL

UpdatePotGrid()

Método das Diferenças Finitas para BVP's

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

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

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

Condições Fronteira

$$\texttt{pot\_grid[0]} = 0$$

$$\texttt{pot\_grid[ngrid - 1]} = 0$$

POTENCIAL

UpdatePotGrid()

Método das Diferenças Finitas para BVP's

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

```
Vec tri_b (ngrid - 2);
for (int i = 0; i < ngrid - 2; i++)
  tri_b[i] = dens_grid[i + 1]*hgrid*hgrid;

EqSolver Banded(Tri_mat, tri_b);

// Solve the system ...
Vec result;
result = Banded.TridiagonalSolver();
```

```
pot_grid[0] = 0;
pot_grid[ngrid - 1] = 0;
for (int i = 0; i < ngrid - 2; i++){
  pot_grid[i + 1] = result[i];
}
```

CAMPO ELÉTRICO

Spline3Interpolator::Deriv()

Spline Cubic Interpolation

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

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

```
if (x[0] <= fx && fx <= x[N - 1]){
  i = Binary_Search(N, x, fx);
}
...
Derivative = 1./6. * K[i] * (((3.0*pow(fx - x[i + 1], 2)) / (x[i] - x[i + 1])) - (x[i] - x[i + 1])) -
             1./6. * K[i + 1] * (((3.0*pow(fx - x[i], 2)) / (x[i] - x[i + 1])) - (x[i] - x[i + 1])) +
             ((y[i] - y[i + 1]) / (x[i] - x[i + 1]));
```

POSIÇÃO / VELOCIDADE

Sistema de ODE's

$$\begin{cases} \frac{dr_i}{dt} = v_i \\ \frac{dv_i}{dt} = -E(r_i) = \frac{d\phi(r_i)}{dx} \end{cases}$$

TimeStep()

```
void ODEsolver::UpdateParameter(int i, int p, double value){
  F[i].SetParameter(p, value);
} // set parameter 'p' from TFormula 'i' to 'value'
```

```
  Spline3Interpolator Pot (ngrid, xgrid, pot_grid, dens_grid[0], dens_grid[ngrid - 1]);
  // cout << Pot.Deriv(x_min) << " -- E(x_min) ~ E(x_max) -- " << Pot.Deriv(x_max) << endl;


  for (int i = 0; i < npart; i++){
    Equation.UpdateParameter(1, 0, Pot.Deriv(x_vpart[i][0]));
    x_vpart[i] = Equation.RK4_iterator(x_vpart[i], dt);
    // x_vpart[i] = Equation.RK4solver(x_vpart[i], 0.0, dt, dt)[1]; -- Opção mais demorada !

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

CRIAÇÃO E GRAVAÇÃO DE FIGURAS

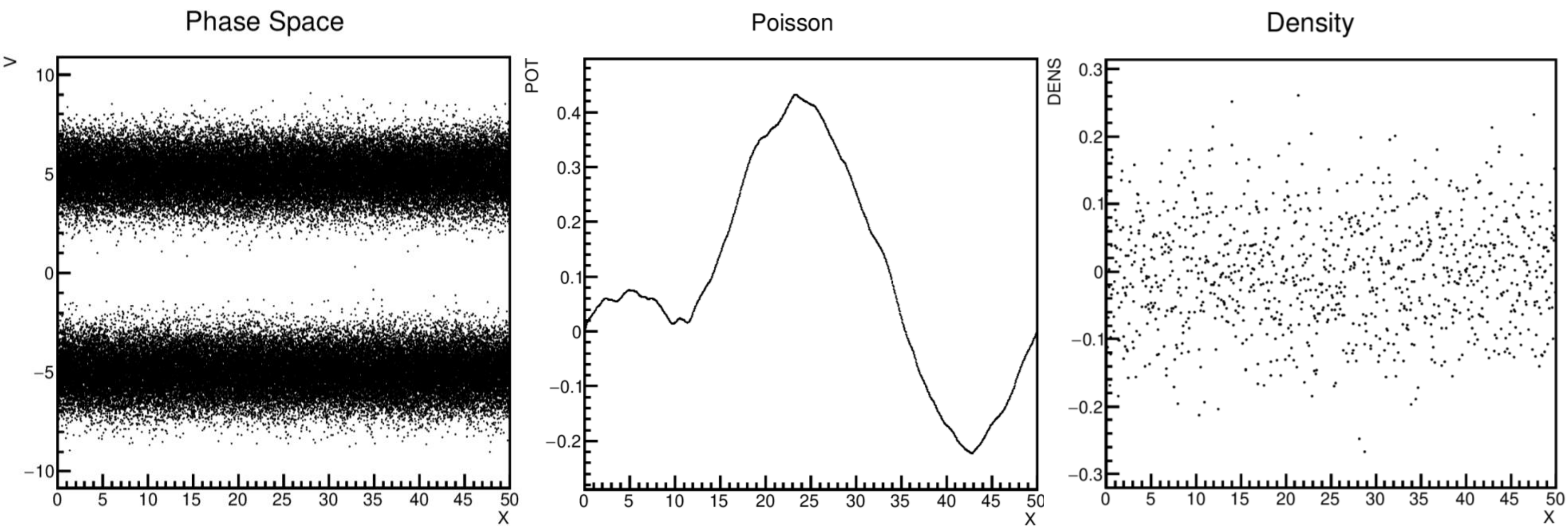
```cpp
void FdistV (vector<double> veloc, bool save_plot, bool show_plot = false);

void Plot_Phase_Space (bool save_plot, bool show_plot = false);

void Density (bool save_plot, bool show_plot = false);

void Poisson (bool save_plot, bool show_plot = false);
```

```cpp
int PIC::nshow = 0;

TCanvas PIC::Save_c ("Save_c", "PIC - Graph", 1500, 500);

TCanvas PIC::Save_dist ("Save_dist", "PIC - Distribution", 1000, 500);

TApplication PIC::App ("PIC", NULL, NULL);

TCanvas PIC::App_c ("App_c", "PIC - Real Time Plotting");
```

```cpp
TGraph phase_space;

TGraph poisson;

TGraph density;


void SetAppCanvas();

void SetGraphs();

void SetSaveCanvas();

void SetSaveDist();
```

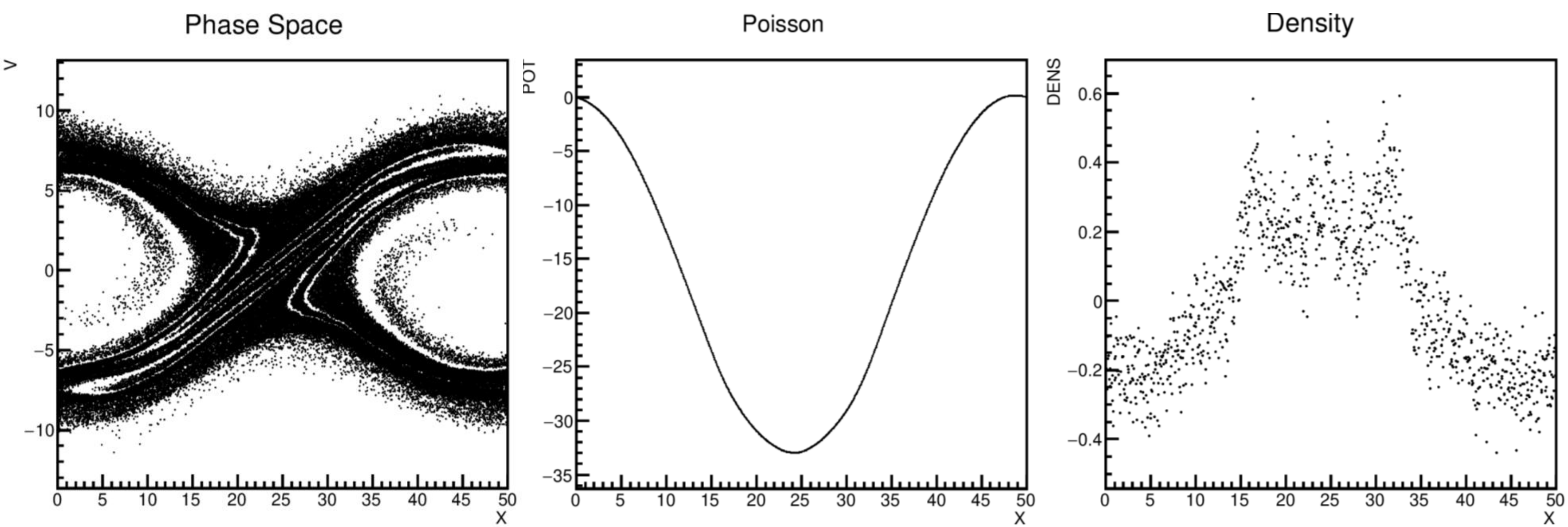
RESULTADOS OBTIDOS

- **Caso (1)** → $t = 0$



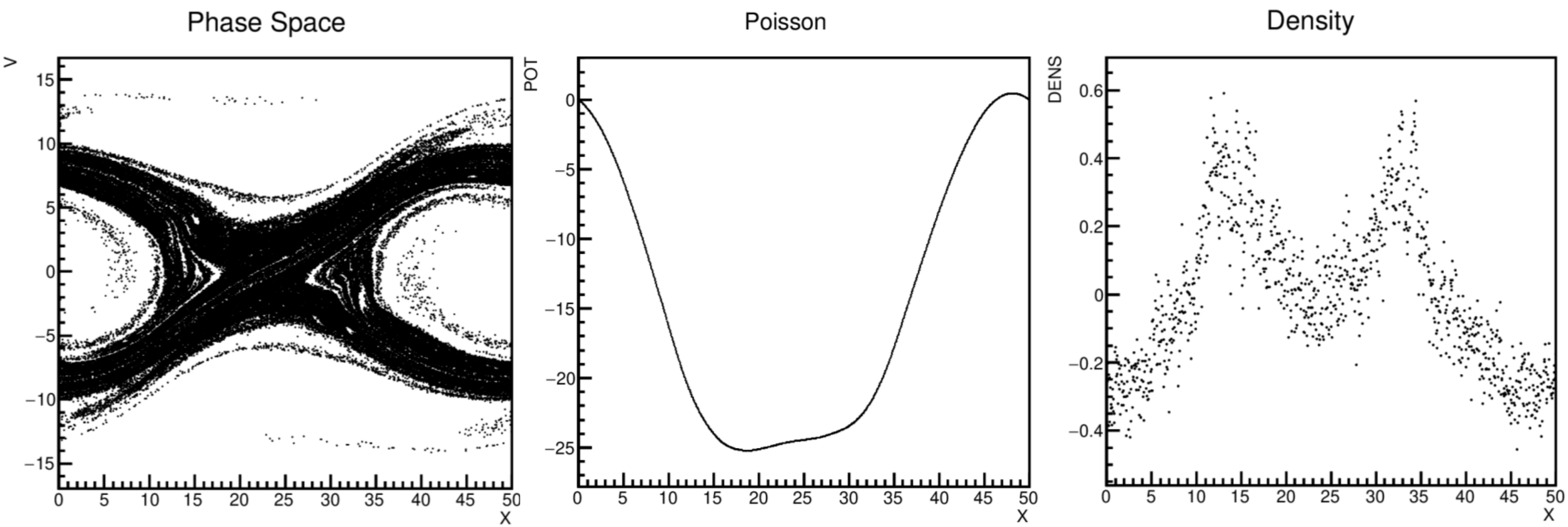
RESULTADOS OBTIDOS

- **<u>Caso (1)</u>** → $t = 32$

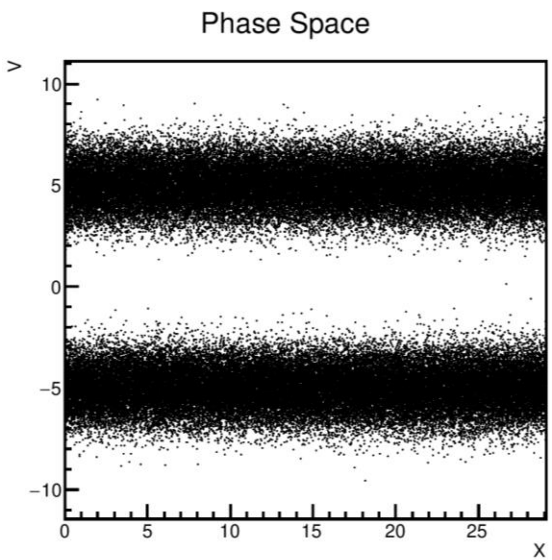


RESULTADOS OBTIDOS

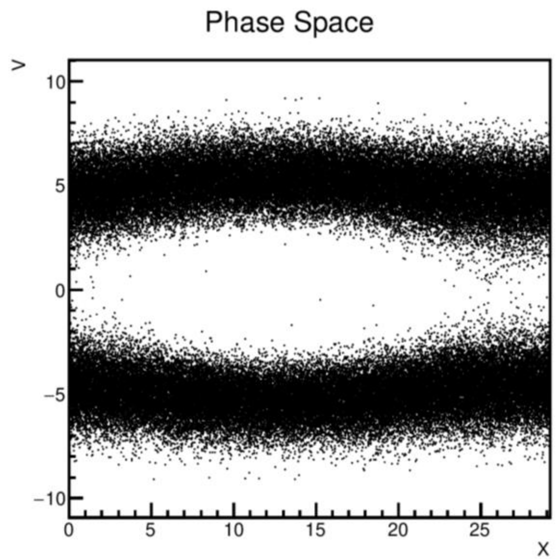
- **Caso (1)** → $t = 60$

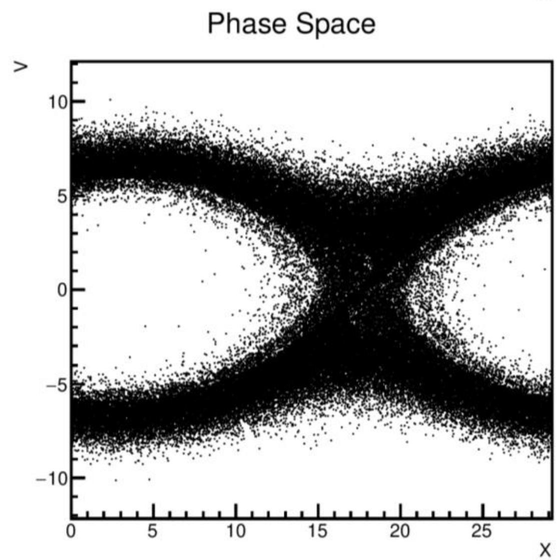


- **Caso (2)**

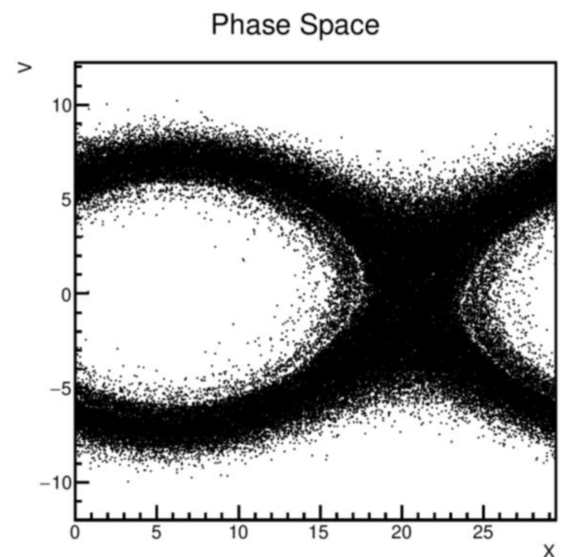


$$\Delta t = 120 \qquad x_{max} = 29.1$$





$$\Delta t = 60 \qquad x_{max} = 29.2$$



$$\Delta t = 60 \qquad x_{max} = 29.4$$

PIC