Autómata Celular Para El Estudio De Un Sistema Cuántico 1D

Mateo De Mendoza, Daniel Peralta,
Nicolás Nino, Brayan Pẽna

Departamento de Física
Universidad Nacional de Colombia
Noviembre 29 de 2022

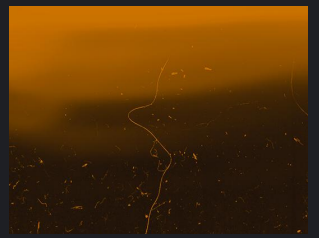
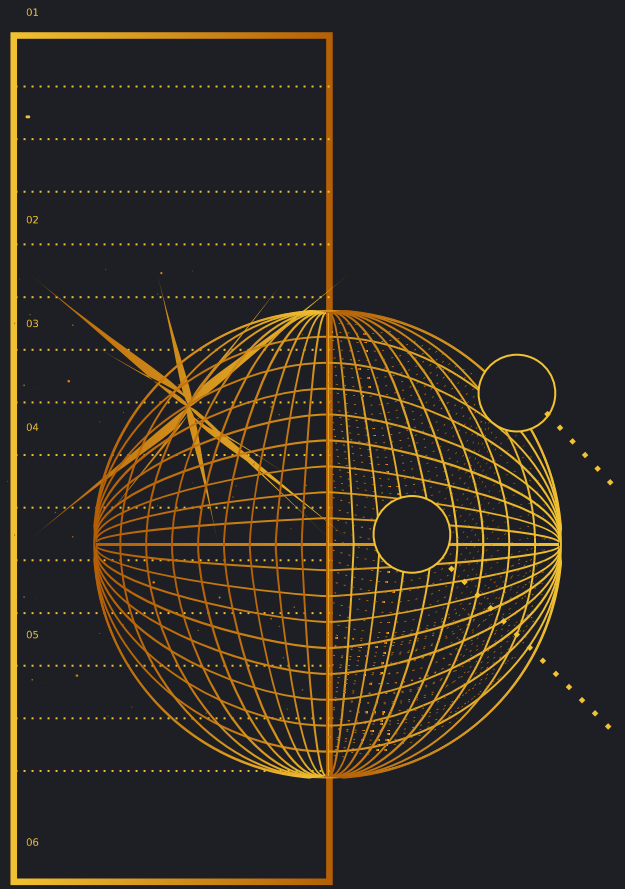
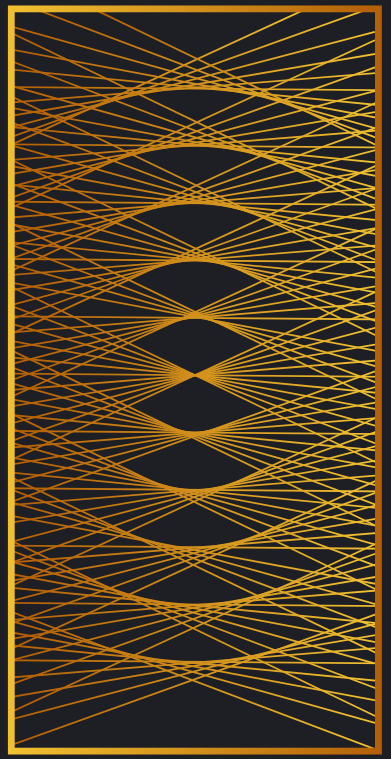
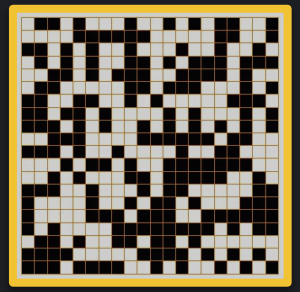


Tabla de Contenido







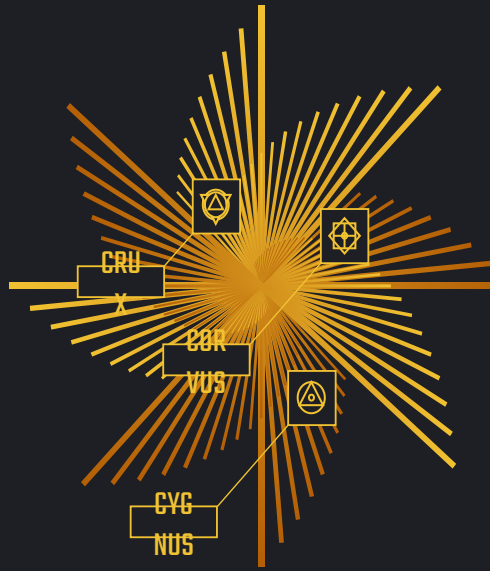
01.
AUTÓMATA CELULAR CUÁNTICO

DISCRETIZACIÓN DEL ESPACIO

{EduardoOrtega2004}.

- El espacio se divide en N celdas
- Cada una contiene dos números complejos que representan el estado de la partícula.
- En cada punto del espacio y en cada momento, la partícula puede estar moviéndose hacia la derecha o hacia la izquierda
- probabilidades de amplitud están representadas por

$$\Psi_r(x_i, t) \qquad \Psi_l(x_i, t)$$



$\Psi_r(x_{i-1}, t)$	$\Psi_r(x_i, t)$	$\Psi_r(x_{i+1}, t)$
$\Psi_l(x_{i-1}, t)$	$\Psi_l(x_i, t)$	$\Psi_l(x_{i+1}, t)$

EVOLUCIÓN DEL SISTEMA

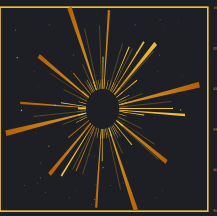
En el esquema de autómatas celulares de difusión, hay dos fases en un solo paso de tiempo, la fase de colisión y la fase de advección. En la fase de colisión tenemos la siguiente regla para el intercambio de contenido local

$$\psi_r(x, t + \Delta t/2) = p\psi_r(x, t) + q\psi_l(x, t))$$
$$\psi_l(x, t + \Delta t/2) = q\psi_r(x, t) + p\psi_l(x, t))$$

Donde p, q ∈ C.

En la fase de advección, los contenidos en cada celda son desplazados a sus primeros vecinos. Para preservar condición la de evolución, las amplitudes p y q deben satisfacer:

$$|p|^2 + |q|^2 = 1 \qquad p^*q + pq^* = 0$$



EVOLUCIÓN DE PARTÍCULA LIBRE

La evolución del estado del sistema completo se expresa como una matriz

$$|\Psi(t+\Delta t)>=\mathbb{MC}|\Psi(t)>$$

$\mathbb{C}$ Fase de Colisión

$\mathbb{M}$ Fase de Advención

$\mathbb{U}=\mathbb{MC}$ = $e^{-i\hat{T}\Delta t}$ Operador temporal sin el potencial

INTRODUCCIÓN DEL POTENCIAL

- Operador de evolución temporal con potencial $\hat{U} = e^{-i\hat{H}\Delta t}$, Donde $\hat{H} = \frac{\hat{P}^2}{2m} + \hat{V}$
- Aproximación de Trotter-Susuki $e^{-i(\hat{T}+\hat{V})t} = \lim_{\tau\to\infty}\left(e^{-i\hat{T}t/\tau}e^{-i\hat{V}t/\tau}\right)^{\tau}$

En términos de simulación, aplicar el operador $\hat{U}$ es equivalente a aplicar muchas veces el operador $e^{-i\hat{T}\Delta t}e^{-i\hat{V}\Delta t}$ con $\tau = t/\Delta t$

- El problema de la introducción potencial se reduce a encontrar la representación matricial.
- Restringir los potenciales a potenciales independientes del tiempo.

- Elementos de matriz son:

$$< x_m, i_\alpha | e^{-i\hat{V}(x)\Delta t} | x_n, i_\beta >= e^{-iV(x)\Delta t}\delta_{m,n}\delta_{\alpha,\beta}$$

Sea $\mathbb{V}$ la matriz del potencial, entonces la evolución del sistema total es

$$|\Psi(t+\Delta t)>= \mathbb{VMC}|\Psi(t)>$$

Las reglas de evolución son:

$$\psi_r(x+\Delta x, t+\Delta t) = e^{-iV(x)\Delta t}(p\psi_r(x,t)+q\psi_l(x,t))$$
$$\psi_l(x-\Delta x, t+\Delta t) = e^{-iV(x)\Delta t}(p\psi_l(x,t)+q\psi_r(x,t))$$

CLASE LATTICE BOLTZMANN CUÁNTICO

Atributos , Métodos y más

```cpp
#include <cmath>
#include <complex> //Standard c++ library for complex numbers.
#include <iomanip>
#include <iostream>
#include <eigen3/Eigen/Dense>

//alias for complex numbers
typedef std::complex<double> complex;

const int L = 300; // space size
const double theta = M_PI / 4;
complex p(std::cos(theta), 0); // Transition amplitudes
complex q(0, std::sin(theta));
const int Q = 2; // Number of directions
const int N = Q*L; //Dimension of the vectors we will be using

// typedef ('aliases')
typedef Eigen::VectorXcd Vector;
typedef Eigen::MatrixXcd Matrix;
```

EIGEN: biblioteca C++ de alto nivel para álgebra lineal, operaciones matriciales y vectoriales , transformaciones geométricas, solucionadores numéricos y algoritmos relacionados.

```cpp
class QLB {
private:
  Vector Psi ; // Wave vectors are created as private attributes.
  Vector Psi_new;
  Matrix M;
  Matrix C;

public:
  QLB(void); //constructor. Initialize state as zero
  void Start(void);     // This imposes the initial conditions.
  void Get_Psi(void);   // Returns the Wave vector. Just for test.
  void Get_Psi_new();   // Returns the Wave vector. Just for test.
  complex Rho(int ix);  // Returns the probability density at each cell.
  void Collision(void); // Creates the Collsion operator and modifies Psi_new.
  void Advection(void); // Creates the Advection operator and modifies Psi.
  void Print_Rho(void); // Prints the real part fo the wave function, later, the
                        // function will pirnt the probability density.
  void M(void); //for testing
```

Clase QLB

Constructor

```
QLB::QLB(void){
  int i,j;
  //Declare the size of the matrices,
  Psi.resize(N);
  Psi_new.resize(N);
  M.resize(N,N);
  C.resize(N,N);
  for (i=0; i<N; i++){
    Psi(i) = (0,0);
    Psi_new(i) = (0,0);
  }
  // initialize the Collision Catrix
  for (j = 0; j<N ; j++){
    //divide into odd and even case
    for(i=0; i<N; i++){
      if(i==j){
        C(i,j)=p;
      }
      else{
        if(j%2 ==0){
          if(j-i == 1){C(i,j)=q;}
          else {C(i,j)=0;}
        }
        else{
          if(i-j == 1){C(i,j)=q;}
          else {C(i,j)=0;}

        }
      }
    }
  }
```

```
  // initialize the Advection
  M = Matrix::Zero(N,N);
  for (int j = 0; j <  N; j++) {
    if (j % 2 == 0) {
      M((j + 2 +  N) % (N),j) = (1, 1);
    } else if (j % 2 == 1) {
      M((j + 2 * L - 2 + N) % (N),j) = (1, 1);
    }
```

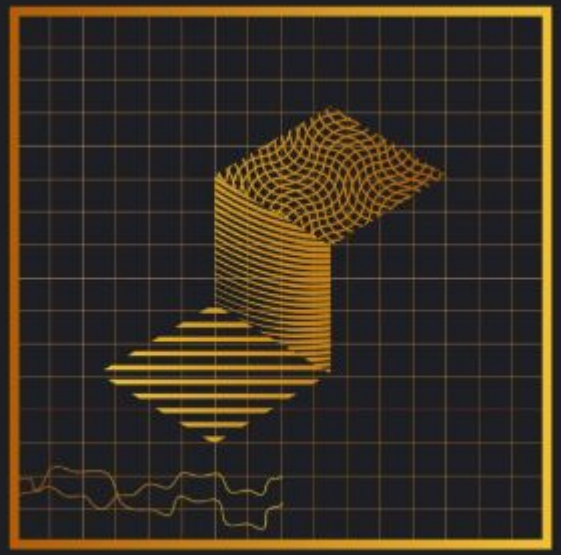
inicialización de la matriz de colisión y advección

```cpp
void QLB::Get_Psi(void) {
  for (int ix = 0; ix < N; ix++)
    std::cout << Psi(ix,0) << std::endl;
}

void QLB::Get_Psi_new(void) {
  for (int ix = 0; ix < N; ix++)
    std::cout << Psi_new(ix,0) << std::endl;
}
complex QLB::Rho(int ix) { return Psi(ix,0) + Psi(ix + 1,0); }

void QLB::Collision(void) {
  Psi_new = C * Psi;
}

void QLB::Advection(void) {
  Psi = M * Psi_new;
}

void QLB::Print_Rho(void) {
  for (int ix = 0; ix < N; ix += 2)
    std::cout << ix/2 << " " << std::norm(Rho(ix)) << std::endl;
    //Add two blank lines for animating in gnuplot
    std::cout << "\n"<< "\n";
}
```

```cpp
void QLB::Start(void) {

  for (int ix=0; ix<N; ix++){
    if (ix % 2 == 1) {
      double n=10;
      double k =(2*n*M_PI)/N;
      double sigma0 = 15;
      double argument = -std::pow( (ix/2 - L/2.0)/sigma0, 2);
      double gaussian;
      gaussian = N* 1/(std::pow(2*sigma0*sigma0*M_PI,0.25))*std::exp(argument);
      complex z (std::cos(k*ix)*gaussian, std::sin(k*ix)*gaussian);
      Psi(ix) = z;
    }
  }
```

```cpp
int main() {
  std::cout << std::fixed
            << std::setprecision(
                 3); // This is to choose the precision of complex numl
  QLB free_particle;
  free_particle.Start();
  for (int t = 0; t < 500; t++) {
    free_particle.Print_Rho();
    free_particle.Collision();
    free_particle.Advection();
  }

  return 0;
}
```

Métodos y main



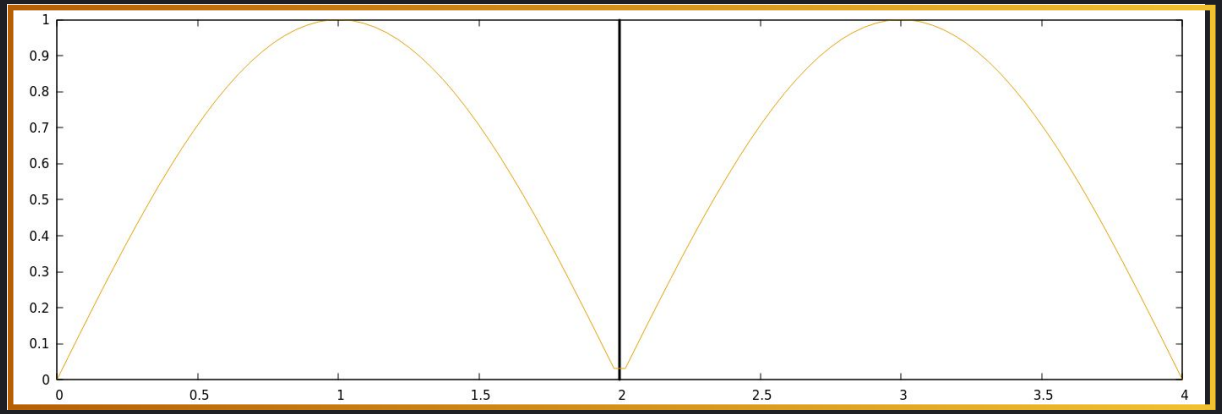
03.
PARTICULA LIBRE

Onda Plana

Relación de dispersión

Condiciones Iniciales de onda Plana

Para imponer condiciones iniciales de onda plana se debieron tener en cuenta las condiciones de fronteras periódicas



Condiciones Iniciales de onda Plana

Se usaron dos ondas iniciales distintas con diferentes números de onda que cumplieran esta condición

$$k = \frac{2\pi n}{L}$$

```
void QLB::Start(void){
  double k = ( 2* M_PI / L);
  double mu = L/2.0;
  for (int ix = 0; ix < Q * L; ix++) {
    if (ix % 2 == 1) {
      complex z(std::cos(k * ix)/2.0, -1 * std::sin(k * ix)/2.0);
      Psi(ix) = z;
    }
  }
}
```

Relación de dispersión de De Broglie

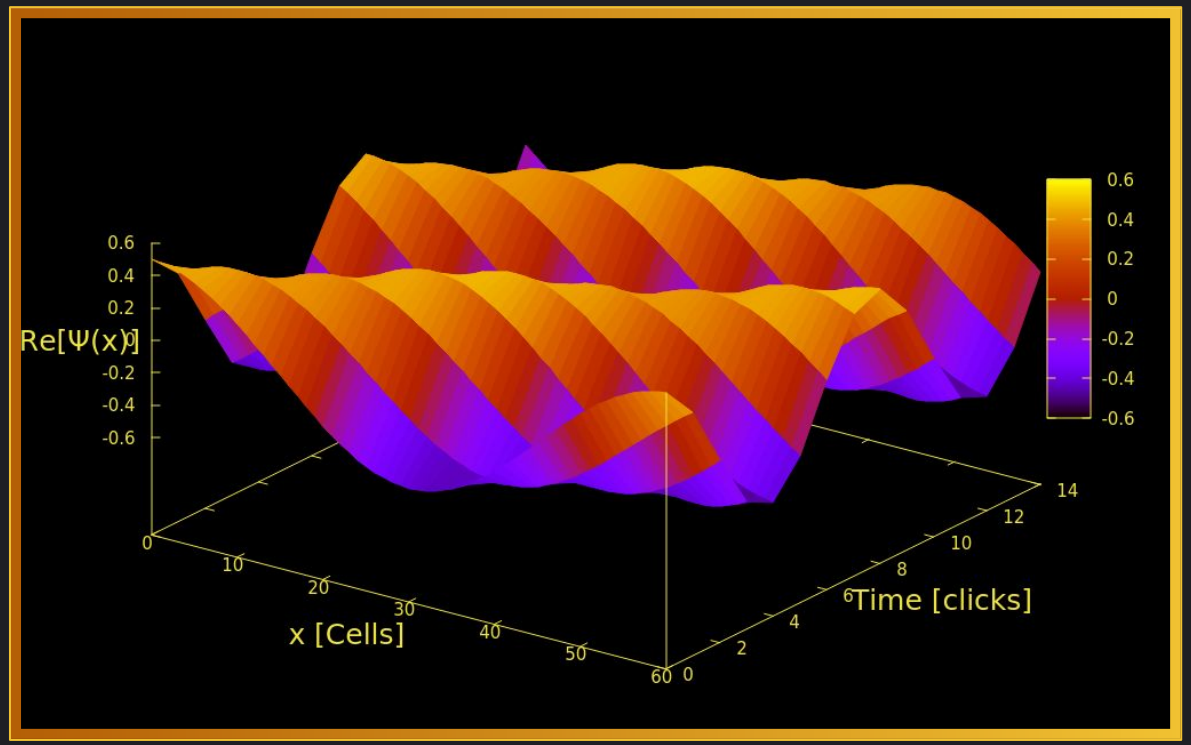
Por lo tanto, el periodo espacial de la función de onda debe ser un divisor del ancho del espacio de simulación.

$$k_1 = \frac{2\pi}{L} \quad k_2 = \frac{4\pi}{L}$$

Del principio de DeBroglie y de la relación entre la frecuencia y la energía de Planck se deriva la relación de dispersión de DeBroglie.

$$\omega = \frac{\hbar k^2}{2m}$$

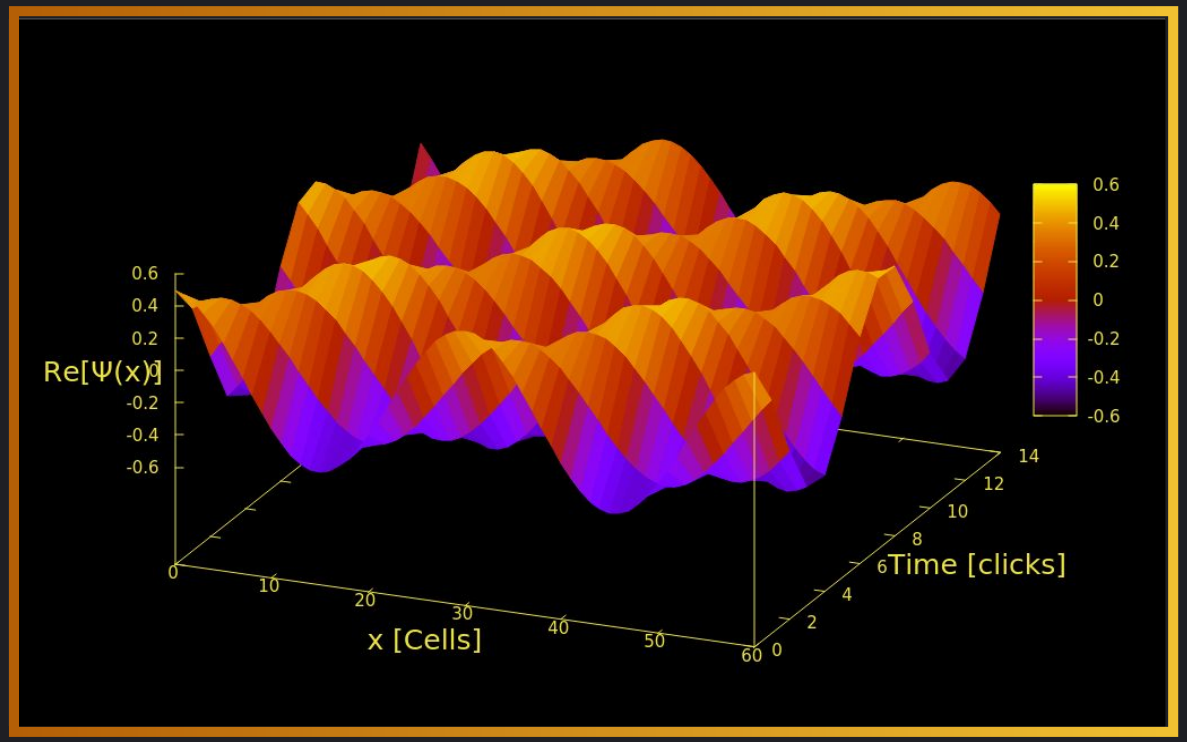
Relación de Dispersión



$$k_1 = \frac{2\pi}{L}$$

No se observa un cambio considerable en el periodo temporal que deberia presentarse debido al cambio en el número de onda

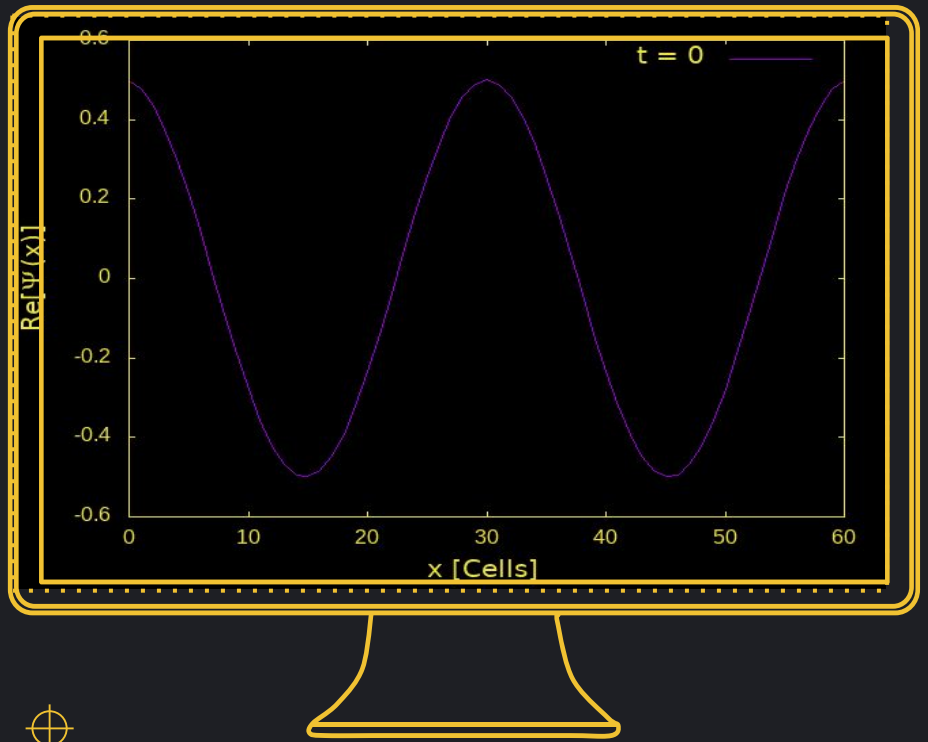
Relación de Dispersión

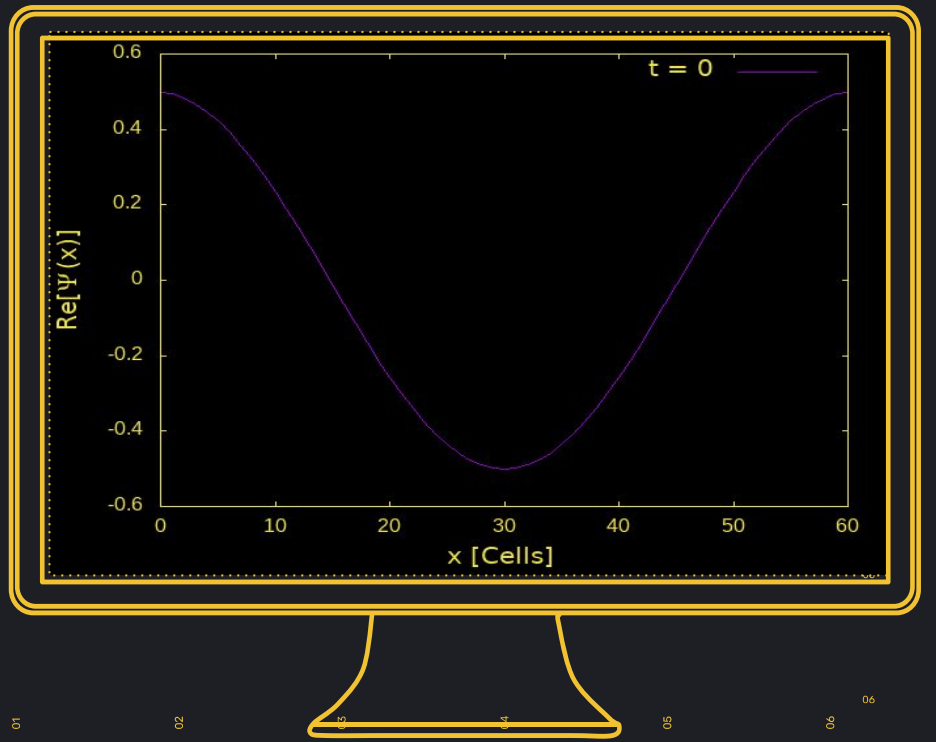


$$k_2 = \frac{4\pi}{L}$$

No se observa un cambio considerable en el periodo temporal que deberia presentarse debido al cambio en el número de onda

Evolución Temporal de las ondas





Paquete Gaussiano

Estudio de la varianza y de otros efectos no físicos.

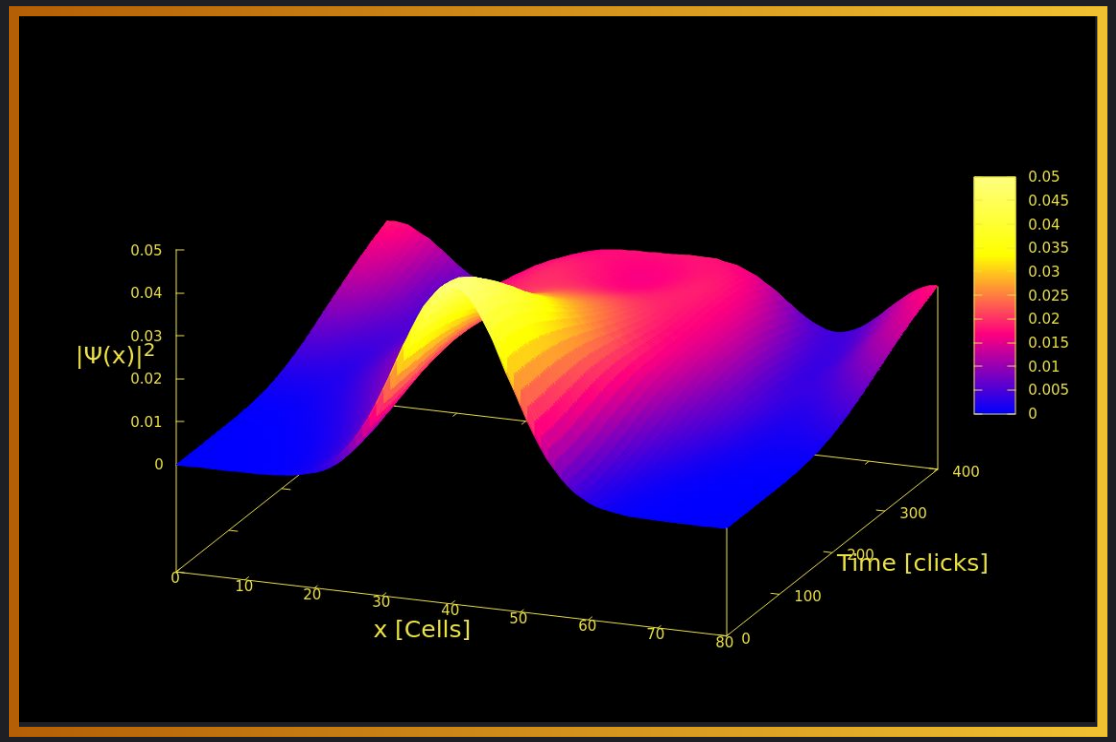
Condiciones Iniciales del paquete gaussiano

Se corrió el programa con dos distribuciones iniciales gaussianas

$$\sigma_1^2 = \frac{L^2}{100}$$

```
const double sigma2 = L*L/100.0;
void QLB::Start(void){
  double k = ( 2* M_PI / L);
  double mu = L/2.0;
  for (int ix = 0; ix < Q * L; ix++) {
    if (ix % 2 == 1) {
      Psi(ix) =  std::exp(-std::pow(ix/2.0 - mu, 2)/(4*sigma2))/std::pow(sigma2*2*M_PI, 0.25);
    }
  }
```

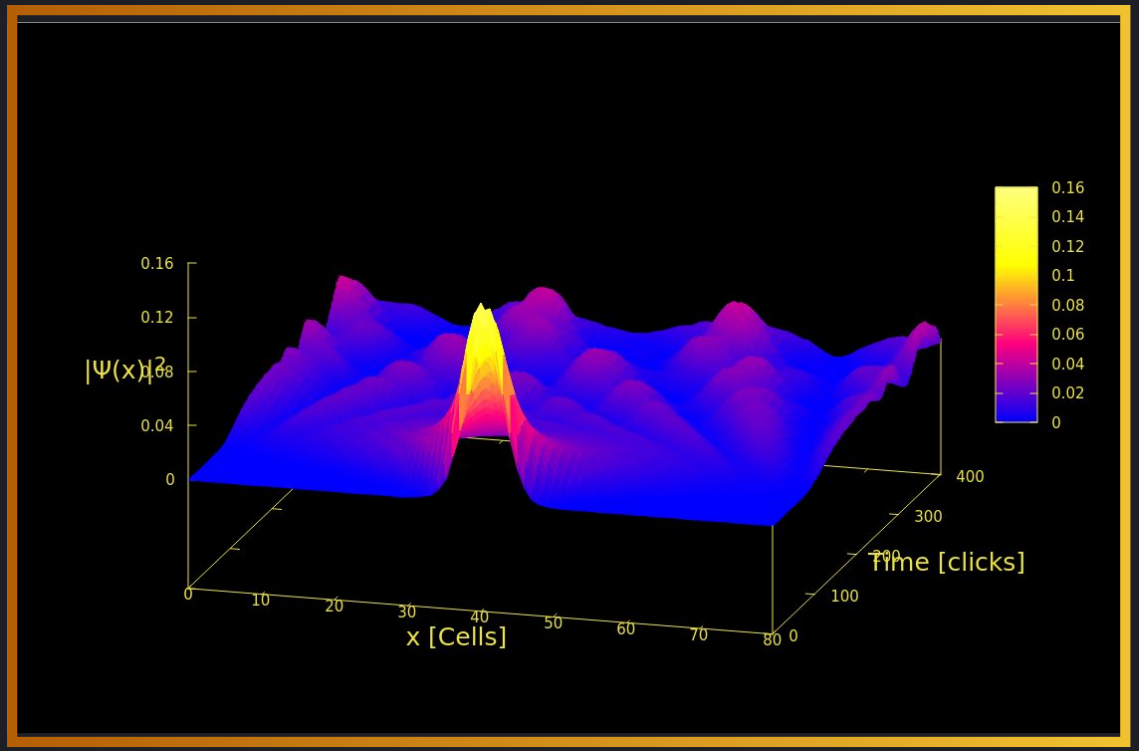
Efecto de la suavidad de la distribución



$$\sigma_1^2 = \frac{L^2}{100}$$

Notamos que entre menos suave sea la distribución más brusca es la oscilación y efectos no físicos

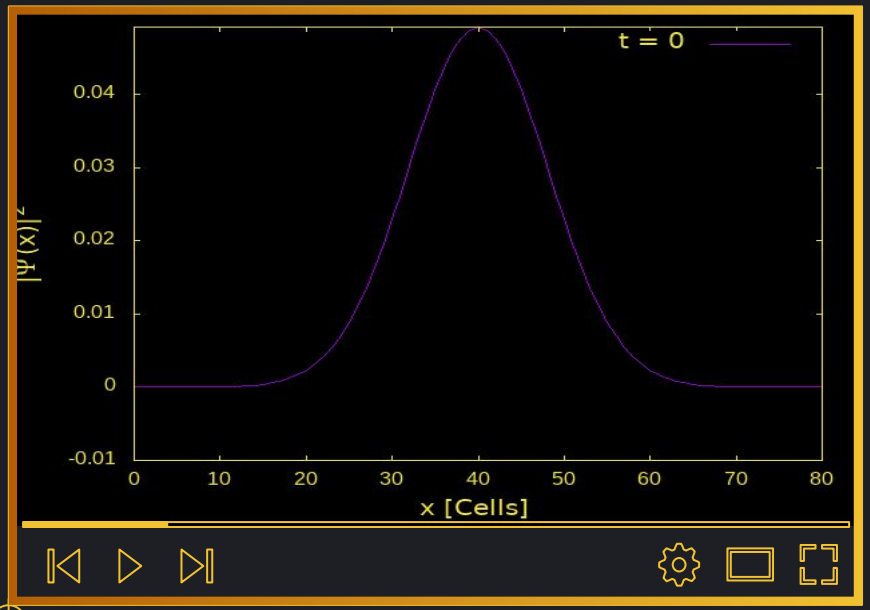
Efecto de la suavidad de la distribución

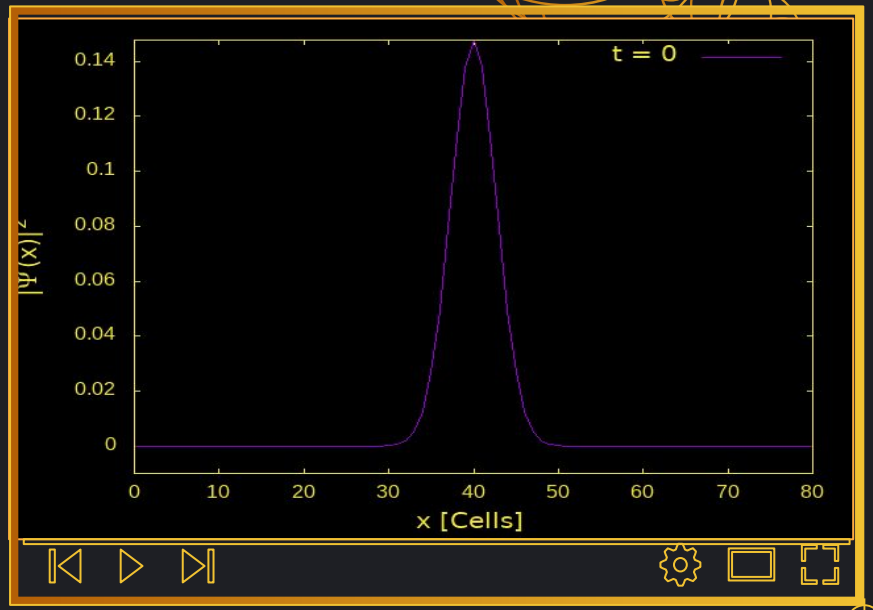


$$\sigma_2^2 = \frac{L^2}{900}$$

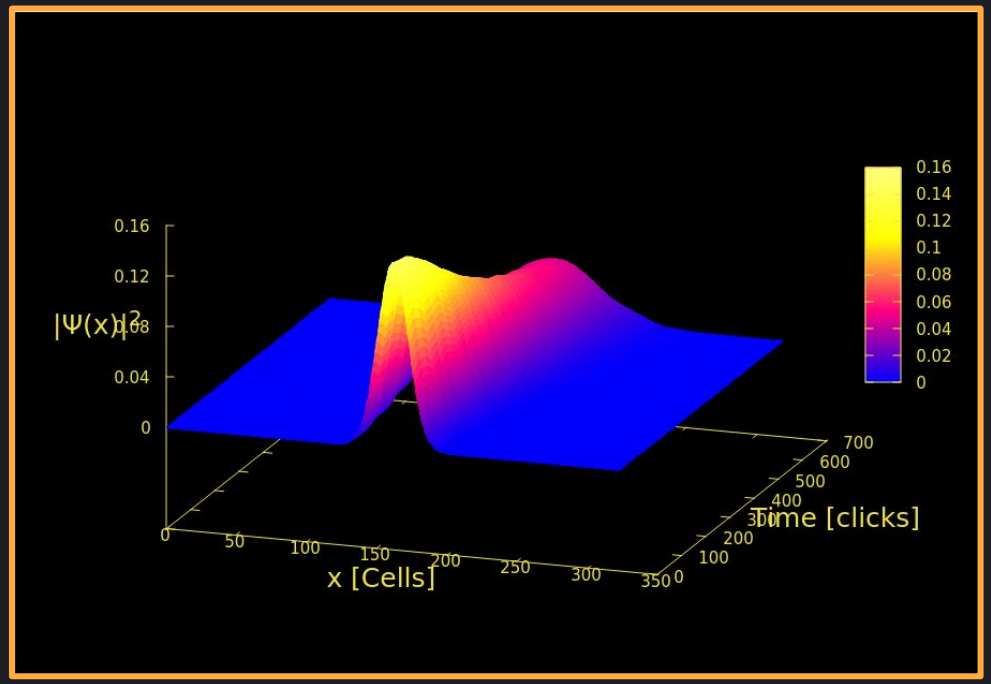
Notamos que entre menos suave sea la distribución más brusca es la oscilación y efectos no físicos

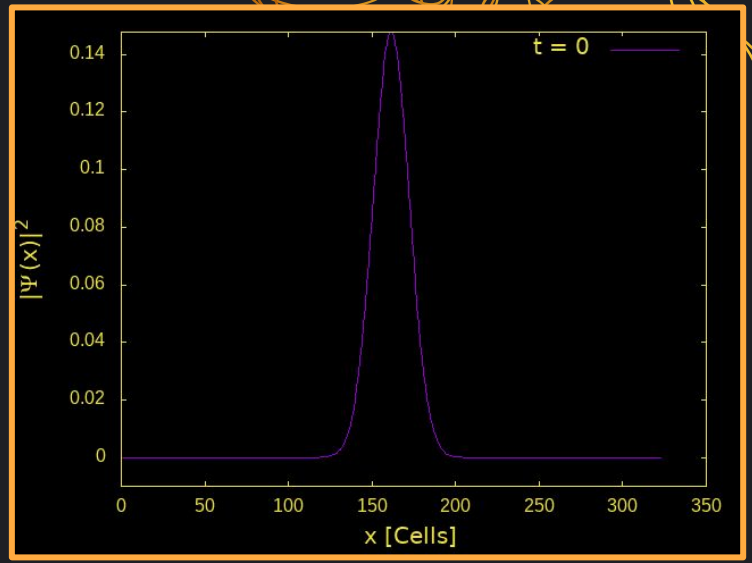
Evolución temporal de la gaussiana y aparición de los efectos no físicos





Aumento en la resolución del Lattice



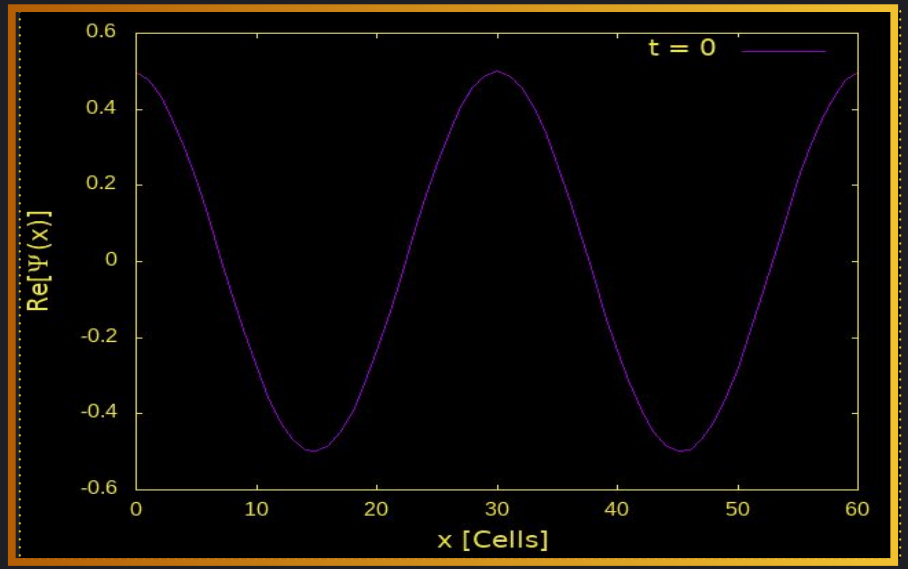


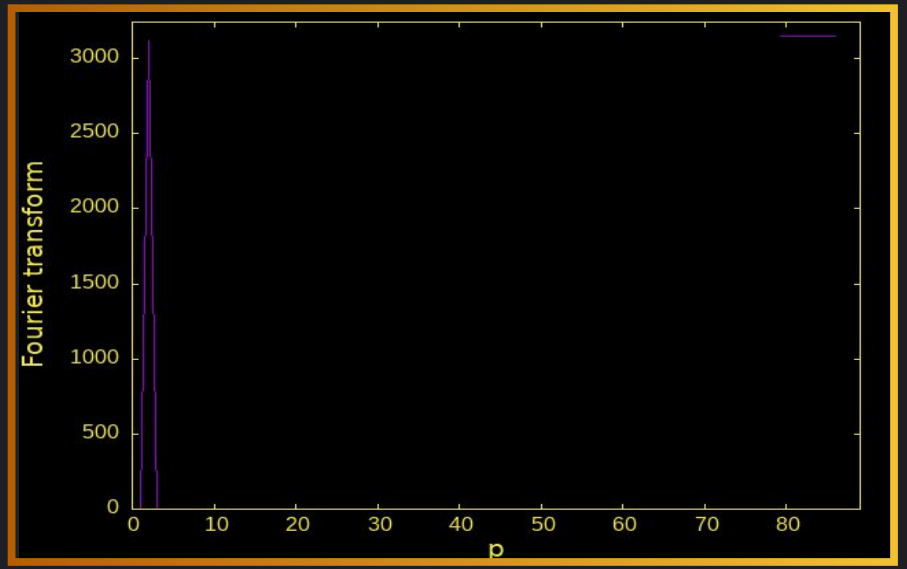
Transformadas de Fourier

$$\{\mathbf{x_n}\} \rightarrow \{\mathbf{X_k}\} \qquad X_k = \sum_{n=0}^{N-1} x_n \exp(-\frac{i2\pi}{N} kn)$$

```
void QLB::DFT(void){

  complex lenght(1/L,0);
  Vector AUX;
  AUX.resize(L);
  for(int ix = 0; ix<L; ix++){

    AUX(ix) = Psi(2*ix) + Psi(2*ix +1);
  }

 for(int k=0; k<L; k++){
   complex sum;
   sum = 0;
   for(int n=0; n<L; n++){
     complex phase(cos((2*M_PI/L)*k*n), -sin((2*M_PI/L)*k*n));
     sum += AUX(n)*phase;

   }
   Phi[k]=sum;
  }
}
```

Transformada de Fourier: Onda Plana





Transformada de Fourier: Gaussiana Viajera

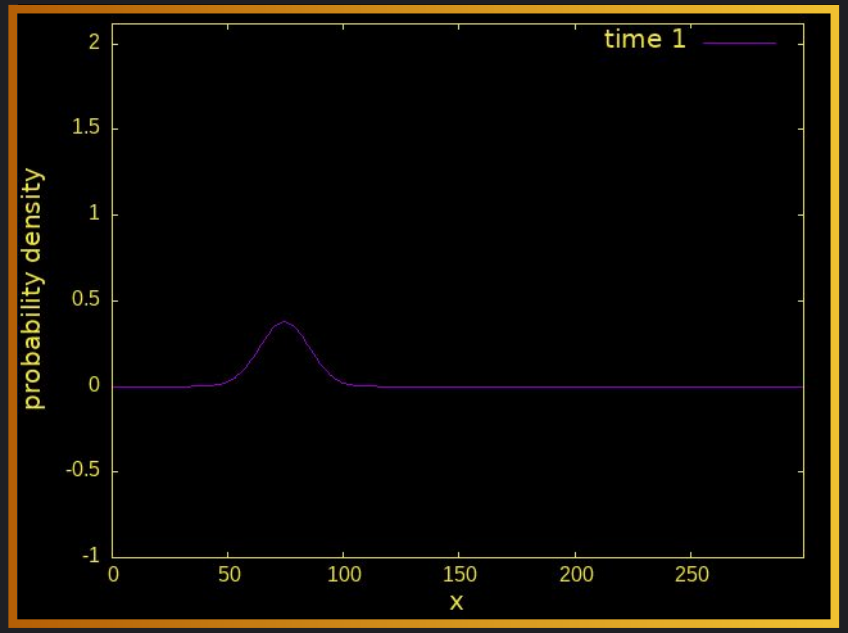
$$\psi(x,t) = \left(\sqrt{2\pi\sigma_x^2}\left(1+\frac{i\hbar t}{2m\sigma_x^2}\right)\right)^{-\frac{1}{2}} \exp\left(-\frac{(x-\frac{p_0}{m}t)^2}{4\sigma_x^2(1+\frac{i\hbar t}{2m\sigma_x^2})}\right)\exp\left(\frac{i}{\hbar}p_0 x\right)\exp\left(-\frac{i}{\hbar}\frac{p_0^2}{2m}t\right)$$

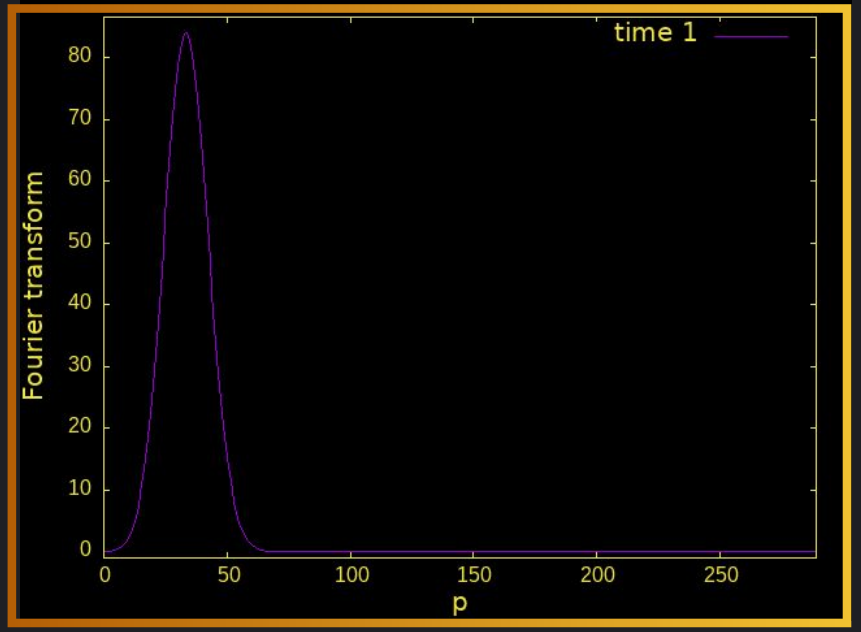
$$\psi(p,t) = \left(\frac{1}{2\pi\sigma_p^2}\right)^{\frac{1}{4}} \exp\left(-\frac{(p-p_0)^2}{4\sigma_p^2}\right)\exp\left(-\frac{i}{\hbar}\frac{p^2}{2m}t\right)$$

De la forma de las funciones de onda se ve claramente que:

$$\bar{p}_t = p_0 \qquad (\Delta p)_t^2 = (\Delta p)_0^2 = \sigma_p^2$$

Transformada de Fourier: Gaussiana Viajera





Varianza de un paquete a través del tiempo

$$\sigma^2(t) = \sigma_0^2 + \frac{\hbar^2}{4m^2\sigma_0^2}t^2$$

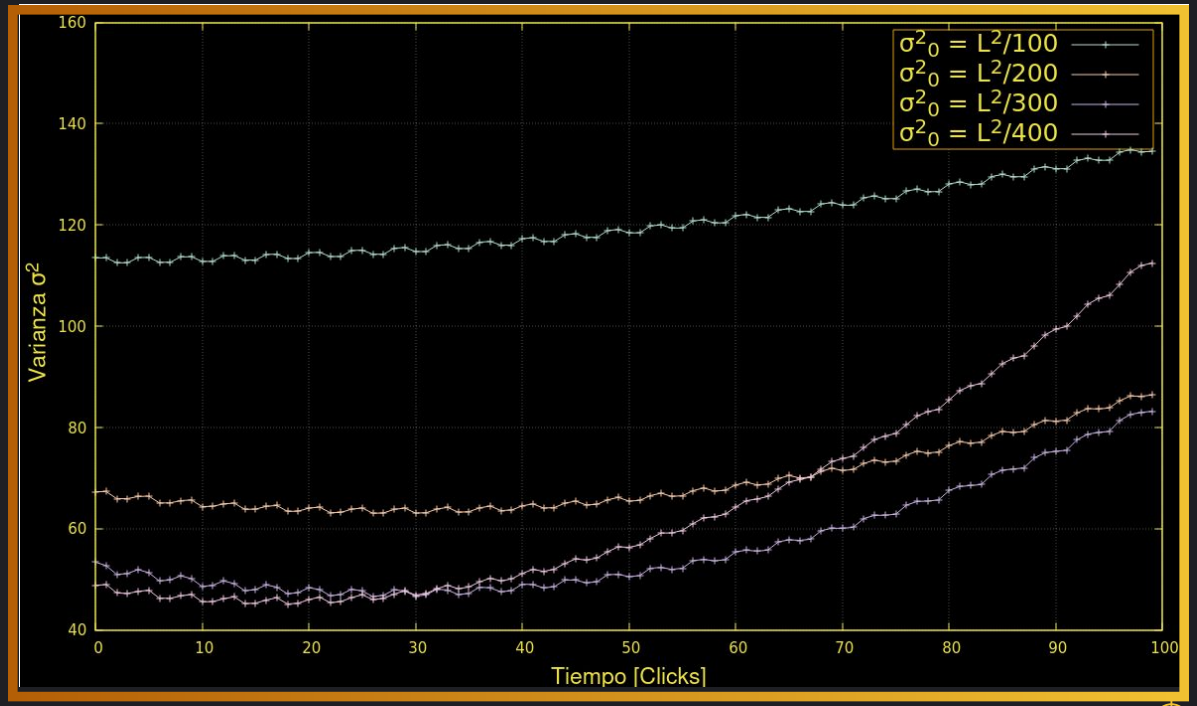
$$\psi(x,t) = \left(\sqrt{2\pi\sigma_x^2}\left(1+\frac{i\hbar t}{2m\sigma_x^2}\right)\right)^{-\frac{1}{2}} \exp\left(-\frac{(x-\frac{p_0}{m}t)^2}{4\sigma_x^2(1+\frac{i\hbar t}{2m\sigma_x^2})}\right)\exp\left(\frac{i}{\hbar}p_0 x\right)\exp\left(-\frac{i}{\hbar}\frac{p_0^2}{2m}t\right)$$

Varianza de un paquete a través del tiempo

$$\sigma^2(t) = a + bt^2$$

$$L = 500$$

$$\theta = \frac{\pi}{4}$$



Varianza de un paquete a través del tiempo

$$b = \tilde{a}x^{\tilde{b}}$$

$$\tilde{b} = -1.35$$

$$\tilde{a} = 12.5$$

Promedio de un paquete Gaussiano

De las expresiones anteriores se tiene:

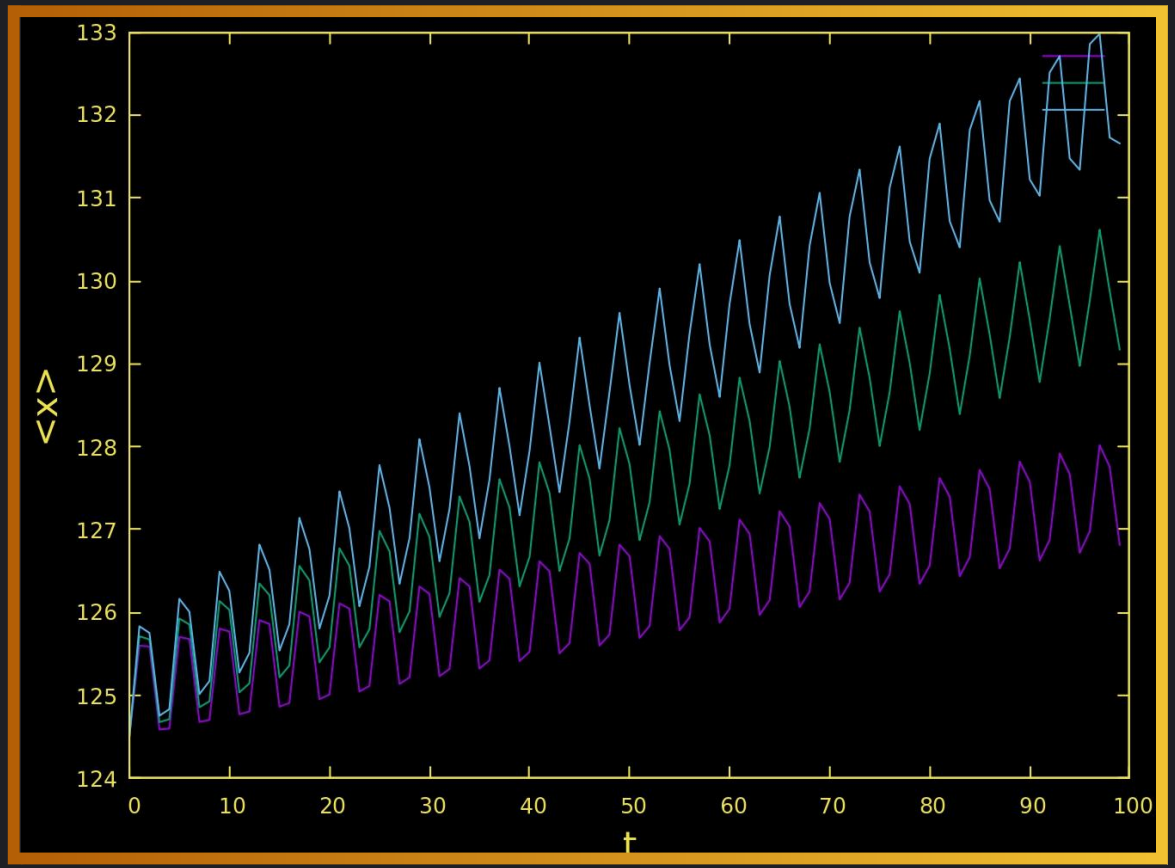
$$\bar{x}_t = \bar{x}_0 + \frac{p_0}{m}t$$

En términos de nuestra simulación:

$$\tan\frac{\pi}{4} = m = 1 \qquad \hbar = 1 \qquad p_0 = k_n = \frac{2\pi n}{500}$$

```
//----------------------------Gaussian-------------------------------
int n = 8;
double k = ( n*2*M_PI / L);
// double k = 0;
double mu = L/2;
double sigma2 = (L*L)/(100);
for (int ix = 0; ix < Q * L; ix++) {
  if (ix % 2 == 1) {
    complex z(std::cos(k * ix), -1 * std::sin(k * ix));
    Psi(ix) = z*std::exp(-std::pow(ix-mu, 2)/(2*sigma2));
  }
  }
```

Promedio de un paquete Gaussiano en el tiempo

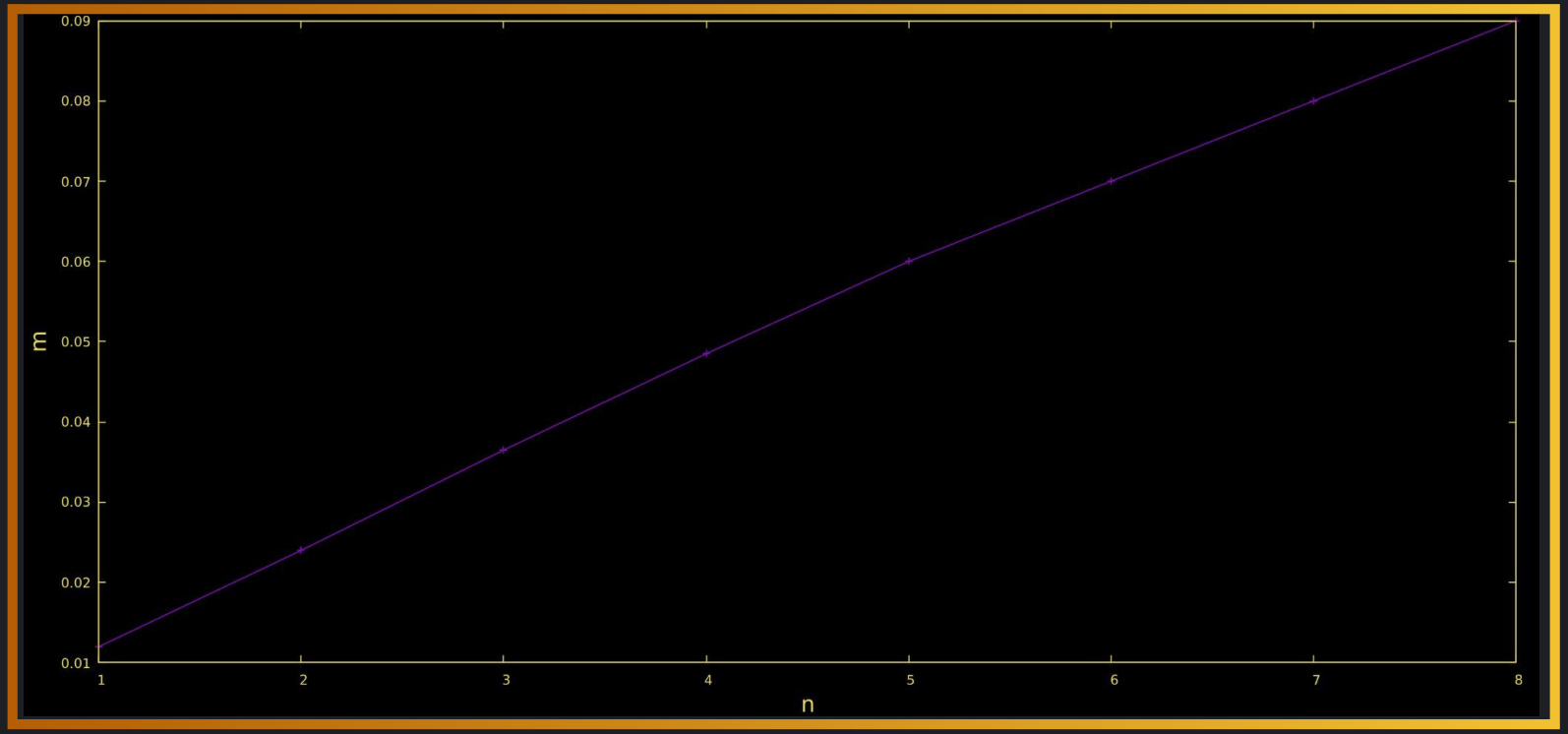


$n = 1$

$n = 2$

$n = 3$

Promedio de un paquete Gaussiano en el tiempo



Promedio de un paquete Gaussiano en el tiempo

Pendiente teórica:

$$m = \frac{2\pi}{500} \approx 0.0125$$

Pendiente medida:

$$m = 0.0112$$

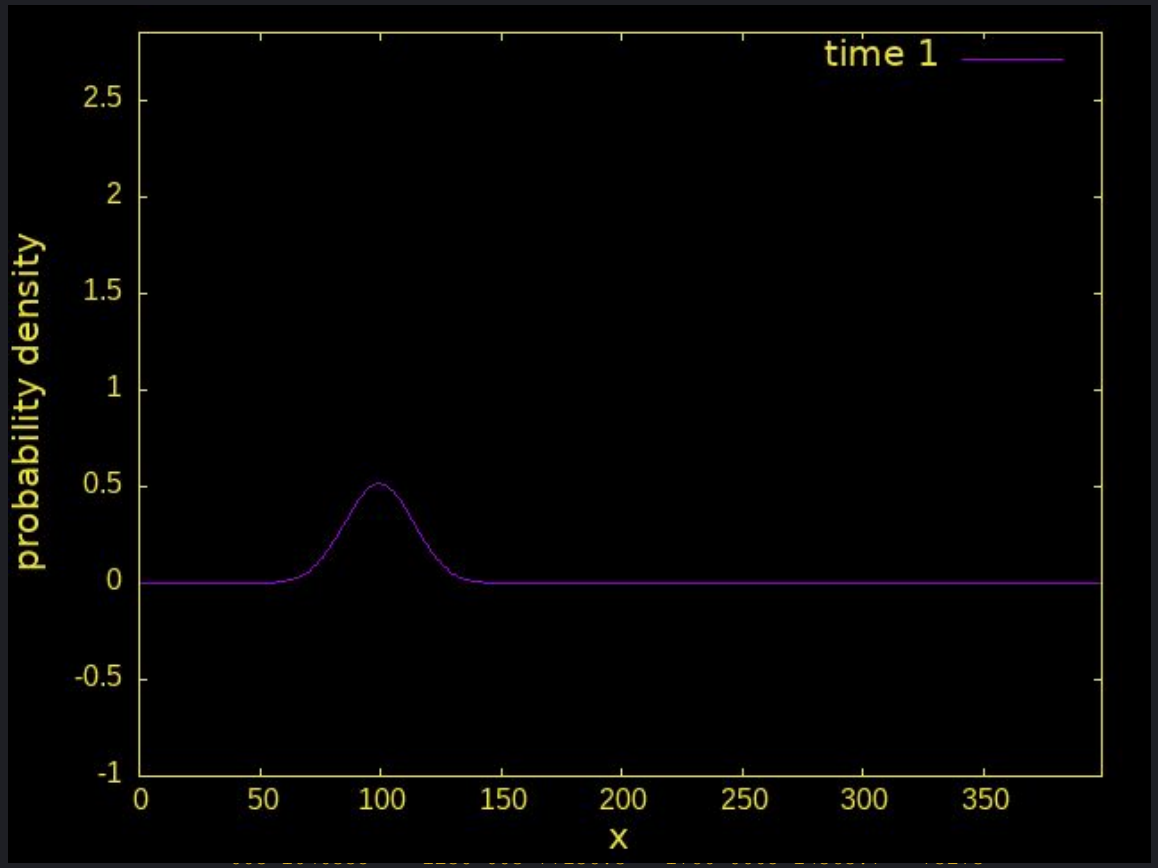
Error: 5%

Introducción del Potencial

```
double epsilon = 1;
 V = Matrix::Zero(N,N);
 for (i=0; i<L; i++){
   double V_x = Potential(i);
   complex aux(std::cos(epsilon*V_x), -std::sin(epsilon*V_x));
   V(2*i+1,2*i+1) = V(2*i,2*i) = aux;

 }
```

```
double Potential(double x){


    if (x<L/2) return 0;
    else{return 10;}

}
```

Pulso Gaussiano con una pared de Potencial



Conclusiones

El método reproduce de manera general el comportamiento del sistema bajo ciertas condiciones iniciales y potenciales específicos. Sin embargo presenta bastantes fallos y aparición de efectos no físicos cuando se va a realizar un estudio más detallado de la física que se pretende representar. Este tipo de errores puede deberse a que no se tiene una calibración adecuada de los parámetros de la simulación.

003-1040559 1250 003-77156.8 1760 0009-14563.7 73273

Gracias

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**