

Moto di Particelle in Campo Elettrico e Magnetico

Pietro Sillano

February 3, 2022

Indice

1	Introduzione	1
2	Scelta dell'integratore numerico	3
3	Singola particella e B	3
3.1	Velocità parallela	3
3.2	Velocità perpendicolare	4
3.3	Confronto RK2, RK4 e Boris Pusher	4
4	ExB Drift	6
4.1	Soluzione analitica	6
4.2	Convergenza degli integratori	7
4.3	Soluzione Numerica	8
5	Particelle in prossimità di un punto a X	9
6	Conclusioni	10
	Appendices	11
.1	Boris Pusher Algorithm	11
.2	Implementazione singola particella	11
.3	Implementazione ExB Drift	16
.4	Implementazione 10k particelle	20

1 Introduzione

In numerosi campi della fisica è importante studiare la dinamica delle singole particelle, sono però pochi i casi in cui una soluzione analitica è possibile.

Per esempio, se considero una particella carica in movimento interagisce con il campo elettromagnetico, l'equazione che ne descrive la dinamica è la seguente :

$$\frac{d\mathbf{v}}{dt} = \frac{q}{m}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \quad (1)$$

La soluzione analitica è possibile in pochi casi, in particolare con configurazioni di \mathbf{E} e \mathbf{B} semplici. Risulta quindi fondamentale ricorrere a metodi numerici, in particolare in questo caso integrare numericamente l'equazione del moto (1).

Come primo passo verso la soluzione numerica è la riscrittura di (1) da unità SI in forma adimensionale, in modo da evitare eventuali cancellazioni numeriche dovute alle differenti scale delle grandezze presenti.

$$\tilde{\mathbf{v}} = \frac{\mathbf{v}}{v_0} \quad \tilde{t} = \frac{t}{t_0} \quad \tilde{\mathbf{E}} = \frac{\mathbf{E}}{E_0} \quad \tilde{\mathbf{B}} = \frac{\mathbf{B}}{B_0} \quad (2)$$

dove v_0 , t_0 , E_0 , B_0 sono costanti con le stesse dimensioni della variabile iniziale in modo da ottenere numeri puri adimensionali. Posso riscrivere \mathbf{v} , t , \mathbf{E} , \mathbf{B} come

$$\mathbf{v} = \tilde{\mathbf{v}} v_0 \quad t = \tilde{t} t_0 \quad \mathbf{E} = \tilde{\mathbf{E}} E_0 \quad \mathbf{B} = \tilde{\mathbf{B}} B_0 \quad (3)$$

Sostituisco nell'equazione (1):

$$\frac{v_0 d\tilde{\mathbf{v}}}{t_0 d\tilde{t}} = \frac{q}{m}(\tilde{\mathbf{E}} E_0 + \tilde{\mathbf{v}} v_0 \times \tilde{\mathbf{B}} B_0) \quad \frac{d\tilde{\mathbf{v}}}{d\tilde{t}} = \frac{qt_0}{mv_0}(\tilde{\mathbf{E}} E_0 + \tilde{\mathbf{v}} v_0 \times \tilde{\mathbf{B}} B_0) \quad (4)$$

$$\frac{d\tilde{\mathbf{v}}}{d\tilde{t}} = \frac{qt_0 E_0}{mv_0} \mathbf{E} + \frac{qt_0}{mv_0} v_0 B_0 \mathbf{v} \times \mathbf{B} \quad (5)$$

$$\frac{qt_0 E_0}{mv_0} = 1 \quad t_0 = \frac{mv_0}{qE_0} \quad (6)$$

sostituisco t_0 nel coefficiente davanti a $\mathbf{v} \times \mathbf{B}$:

$$\frac{qB_0}{m} \frac{mv_0}{qE_0} = \frac{B_0}{E_0} v_0 = 1 \quad E_0 = B_0 v_0 \quad (7)$$

In questo modo ottengo la forma adimensionale dell'equazione di Lorentz (omettendo le tilde):

$$\frac{d\mathbf{v}}{dt} = \mathbf{E} + \mathbf{v} \times \mathbf{B} \quad (8)$$

Non sembra molto diversa rispetto alla prima versione ma ora le grandezze sono adimensionali.

Con questa procedura di adimensionalizzazione definisco delle scale rispetto a cui tratto il problema per via numerica.

$$t_0 = \frac{mv_0}{qE_0} [s] \quad v_0 = \frac{E_0}{B_0} \left[\frac{m}{s} \right] \quad l_0 = v_0 t_0 [m] \quad (9)$$

2 Scelta dell'integratore numerico

Per quanto riguarda la scelta del metodo di integrazione dell'equazione differenziale ho confrontato tre differenti integratori: *Runge-Kutta 2° ordine*, *Runge-Kutta 4° ordine* e in quanto il sistema è hamiltoniano un integratore symplettico che conserva l'energia totale del sistema.

Gli integratori symplettici *Position-Verlet* e *Velocity-Verlet* spesso usati per integrare le equazioni del moto non sono adatti in quanto una delle ipotesi di questi metodi è che la forza sia indipendente dalla velocità.

Ho usato quindi il *Boris Pusher* un integratore symplettico adatto ad integrare l'equazione del moto di una particella soggetta alla forza di Lorentz. (A.1 per l'implementazione)

Per dimostrare i vantaggi dell'integratore di Boris analizzerò un caso semplice ossia il moto di una particella carica immersa in un campo magnetico costante.

3 Singola particella e B

Considero due casi uno in cui la velocità della particella è parallela a B e in cui la velocità è perpendicolare a B.

3.1 Velocità parallela

$$\mathbf{v}_0 = (0, 0, v_z) \quad v_z = 1 \quad (10)$$

$$\mathbf{B} = (0, 0, B_z) \quad B_z = 1 \quad (11)$$

Otteniamo una dinamica di questo tipo:

$$\begin{cases} \frac{dv_x}{dt} = 0 \\ \frac{dv_y}{dt} = 0 \\ \frac{dv_z}{dt} = 0 \end{cases} \quad (12)$$

per cui la \mathbf{v} sarà costante, uguale a v_0 iniziale per cui la particella si muoverà di moto uniforme in direzione z .

3.2 Velocità perpendicolare

$$\mathbf{v}_0 = (v_x, 0, 0) \quad v_x = 1 \quad (13)$$

$$\mathbf{B} = (0, 0, B_z) \quad B_z = 1 \quad (14)$$

$$\begin{cases} \frac{dv_x}{dt} = 0 \\ \frac{dv_y}{dt} = -Bv_x \\ \frac{dv_z}{dt} = 0 \end{cases} \quad (15)$$

Eguaglio la forza di Lorentz con la forza centripeta:

$$F_L = F_c \quad vB = \frac{v^2}{R} \quad B = \frac{v}{R} \quad R = \frac{v}{B} = 1 \quad (16)$$

Calcolo ora il periodo di rotazione sostituendo R:

$$T = \frac{2\pi}{\omega} = \frac{2\pi R}{v} \quad T = \frac{2\pi}{v} \frac{v}{B} = \frac{2\pi}{B} = 2\pi \quad (17)$$

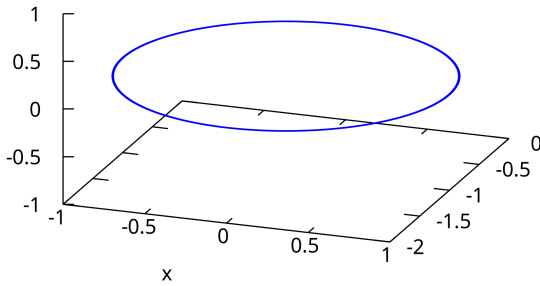
I risultati numerici coincidono con i risultati analitici attesi se si riportano le unità di misura correttamente.

Scegliendo come valori arbitrari $E = 0.01 \frac{V}{m}$ e $B = 10^{-8} T$ e considerando come particella un elettrone quindi $q = 1.6 \times 10^{-19} C$ e $m = 9 \times 10^{-31} kg$ otteniamo le scale rispetto alle quali abbiamo adimensionalizzato il problema:

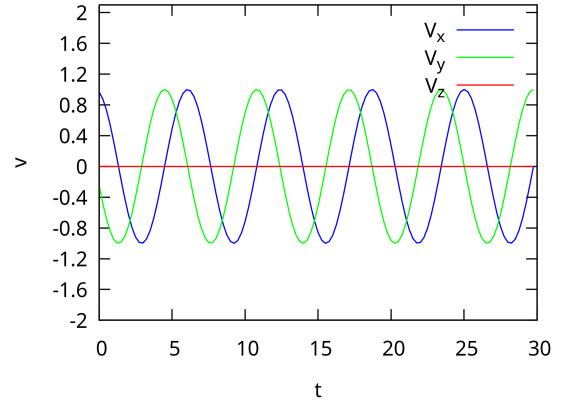
$$t_0 = \frac{mv_0}{qE_0} = 5.625 \times 10^{-4} s \quad l_0 = \frac{E_0 t_0}{B_0} = 562.5 m \quad v_0 = \frac{l_0}{t_0} = 10^6 \frac{m}{s} \quad (18)$$

Per esempio volessimo convertire da grandezza adimensionale a unità SI:

- il raggio dell'orbita circolare: $R = 1 \times l_0 = 562.5 m$
- il periodo dell'orbita circolare: $T = 2\pi \times t_0 = 3.53 s$
- l'energia cinetica dell'elettrone: $E = \frac{1}{2} m \times v_0^2 = 4.5 \times 10^{-19} J$



(a) Traiettoria della particella

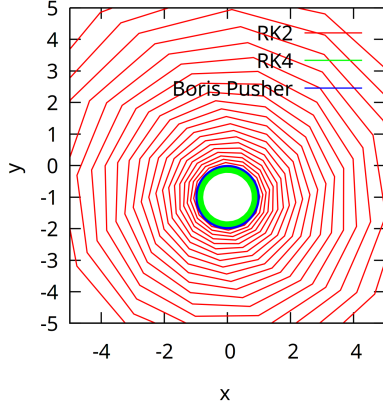


(b) Velocità particella

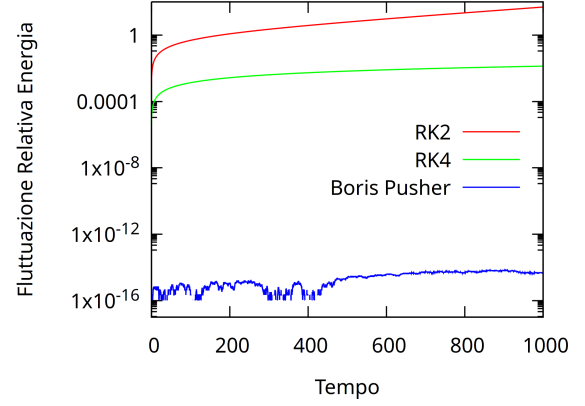
Figure 1: Velocità perpendicolare a \mathbf{B}

3.3 Confronto RK2, RK4 e Boris Pusher

Valuto il diverso comportamento dei tre integratori considerando come varia l'energia della particella durante l'integrazione. Infatti il sistema fisico considerato conserva l'energia iniziale per cui mi aspetto che anche la simulazione numerica la conservi.



(a) Confronto tra le traiettorie



(b) Fluttuazione relativa dell'energia totale

Figure 2: Confronto tra RK2, RK4 e Boris Pusher con un timestep $\Delta t = 0.25$ e 1000 step

$$\Delta E = \frac{|E - E_0|}{E_0} = \frac{|\mathbf{v}^2 - \mathbf{v}_0^2|}{\mathbf{v}_0^2} \quad (19)$$

Valuto l'errore relativo sull'energia considerando soltanto l'energia cinetica della particella in quanto il campo magnetico \mathbf{B} non fornisce energia in quanto è sempre perpendicolare alla direzione di spostamento, influisce quindi soltanto sulla direzione della velocità e non sul modulo. L'errore per RK2 e RK4 raggiunge valori rilevanti e continuerebbe a crescere, mentre l'algoritmo di Boris mantiene un errore trascurabile costante compreso tra 10^{-16} e 10^{-14} . Verifichiamo che l'uso di un integratore simplettico permette la conservazione dell'energia iniziale. Nelle simulazioni seguenti sarà implementato l'integratore di Boris.

4 ExB Drift

4.1 Soluzione analitica

Ora consideriamo il moto di una particella in moto in presenza anche di un campo elettrico \mathbf{E} oltre al campo magnetico \mathbf{B} .

$$\mathbf{B} = (0, 0, B_z) \quad (20)$$

$$\mathbf{E} = (0, E_y, 0) \quad (21)$$

Posso scrivere la \mathbf{v} della particella come somma di una componente parallela e una perpendicolare a \mathbf{B} : $\mathbf{v} = \mathbf{v}_{\parallel} + \mathbf{v}_{\perp}$.

A sua volta $\mathbf{v}_{\parallel} = \mathbf{v}_d + \mathbf{v}_g(t)$ assumendo che la velocità di drift sia costante nel tempo. L'idea è quella di ricondurci a un moto più semplice effettuando un cambio di sistema di riferimento. Infatti, posso

riscrivere (1) come:

$$\frac{d\mathbf{v}_g}{dt} = \mathbf{E} + (\mathbf{v}_d + \mathbf{v}_g) \times \mathbf{B} \quad (22)$$

$$\frac{d\mathbf{v}_g}{dt} = (\mathbf{E} + \mathbf{v}_d \times \mathbf{B}) + \mathbf{v}_g \times \mathbf{B} \quad (23)$$

Posso annullare il termine con il campo elettrico imponendo una determinata \mathbf{v}_d :

$$\mathbf{E} + \mathbf{v}_d \times \mathbf{B} = 0 \quad (24)$$

Applico il prodotto esterno a entrambi i membri e usando un' identità vettoriale, ottengo:

$$\mathbf{v}_d = \frac{\mathbf{E} \times \mathbf{B}}{\|\mathbf{B}\|^2} \quad (25)$$

Mi sono spostato su un sistema di riferimento il cui campo \mathbf{E} é nullo, per cui ritrovo un semplice moto di girazione:

$$\frac{d\mathbf{v}_g}{dt} = \mathbf{v}_g \times \mathbf{B} \quad (26)$$

Se considero $\mathbf{B} = (0, 0, B_z)$ e $\mathbf{E} = (0, E_y, 0)$ avrò:

$$\mathbf{v}_d = \frac{\mathbf{E} \times \mathbf{B}}{\|\mathbf{B}\|^2} = \frac{E_y}{B_z} \vec{i} \quad (27)$$

Posso riscrivere l'equazioni del moto come:

$$\begin{cases} \frac{d(v_x - v_d)}{dt} = v_y B_z \\ \frac{dv_y}{dt} = -(v_x - v_d) B_z + E_y \\ \frac{dv_z}{dt} = 0 \end{cases} \quad (28)$$

La cui soluzione sarà:

$$\begin{cases} v_x = \frac{E_y}{B_z} (1 + \cos(t)) \\ v_y = -\frac{E_y}{B_z} \sin(t) \\ v_z = 0 \end{cases} \quad (29)$$

La traiettoria della particella é una cicloide lungo la direzione x. Il moto è inaspettato avendo un campo elettrico agente lungo y, infatti ci aspetteremmo che la particella acceleri lungo questa direzione.

In realtà, quando la velocità della particella aumenta cresce anche la forza risultante dall'interazione con \mathbf{B} , sempre normale alla traiettoria. Infatti la carica ferma nell'origine subisce un'accelerazione lungo y dovuta a \mathbf{E} . Appena la carica inizia a muoversi la forza magnetica agisce nella direzione normale alla traiettoria. Questo fino alla fine del periodo dove la forza risultante è nulla, per cui la carica si ferma e il moto si ripete periodicamente.

4.2 Convergenza degli integratori

Nel caso precedente di girazione abbiamo studiato le proprietà di conservazione dell'energia dei tre integratori, ora vorremmo studiarne le proprietà di convergenza. Per fare ciò confronterò la soluzione analitica trovata nella sezione precedente con la soluzione numerica. Questo verrà ripetuto per differenti timestep Δt . Come valori per \mathbf{E} , \mathbf{B} e \mathbf{v}_0 della particella useremo:

$$\mathbf{B} = (0, 0, 1) \quad (30)$$

$$\mathbf{E} = (0, \frac{1}{2}, 0) \quad (31)$$

$$\mathbf{v}_0 = (1, 0, 0) \quad (32)$$

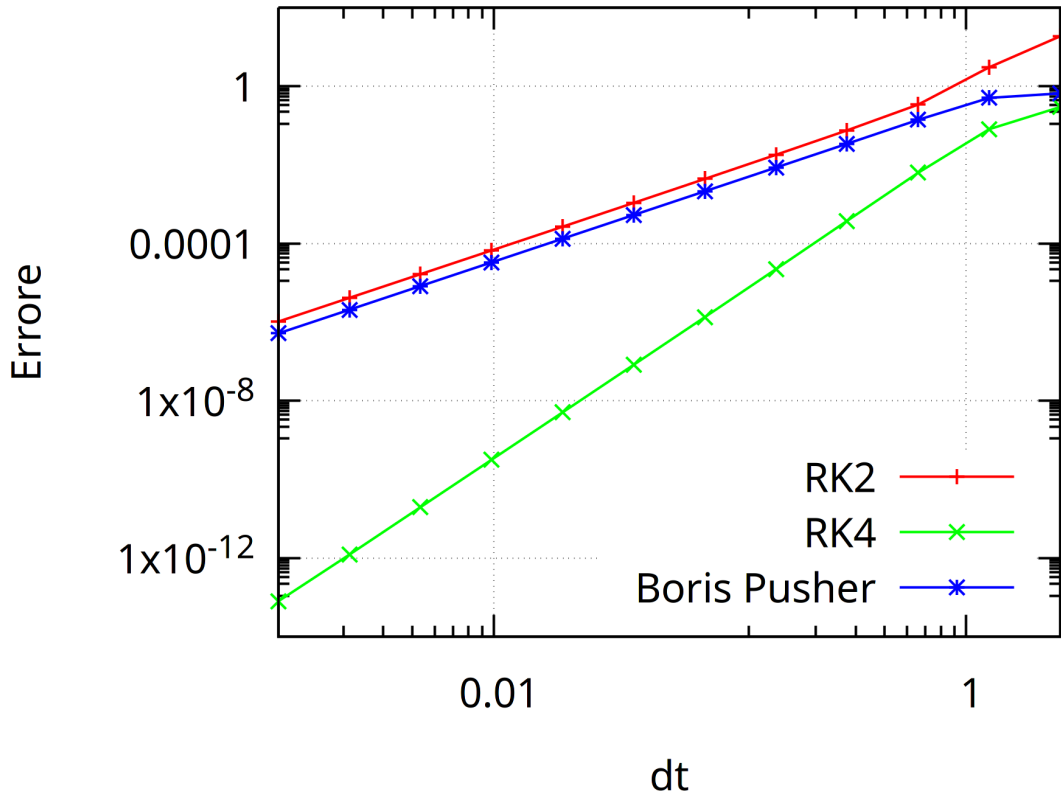


Figure 3: Errore al variare del timestep Δt

Verifichiamo come RK2 e Boris Pusher siano integratori del 2° ordine: un fattore 10 per Δt comporta una riduzione di 100 nell'errore. RK4 è del 4° ordine per cui a un fattore 10 del Δt corrisponde una riduzione di 10^4 per l'errore.

4.3 Soluzione Numerica

Come integratore usiamo Boris Pusher, in quanto nonostante una minore precisione rispetto a RK4 otteniamo un'accuratezza sufficiente: 10^{-4} con timestep $\Delta t = 10^{-2}$.

Come valori per \mathbf{E} , \mathbf{B} e \mathbf{v}_0 della particella useremo:

$$\mathbf{B} = (0, 0, 1) \quad (33)$$

$$\mathbf{E} = (0, \frac{1}{2}, 0) \quad (34)$$

$$\mathbf{v}_0 = (1, 0, 0) \quad (35)$$

Si può notare come la particella non abbia un guadagno netto di energia cinetica. Lo possiamo sp-

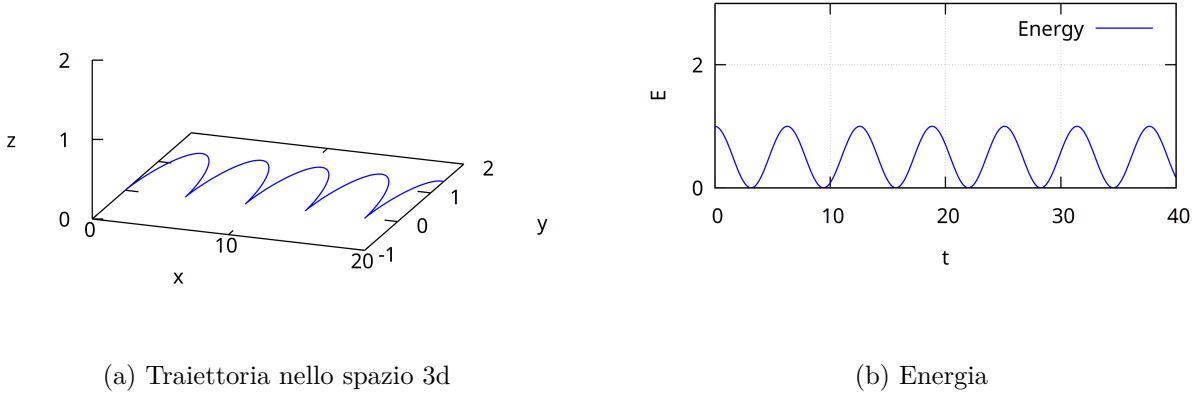


Figure 4: ExB Drift

iegare grazie alla derivazione della sezione precedente: se ci spostiamo in un sistema di riferimento che si muove con \mathbf{v}_d annulliamo il campo \mathbf{E} , per cui la particella interagisce soltanto con \mathbf{B} . \mathbf{B} non compie lavoro sulla particella, infatti non modifica il modulo della velocità ma soltanto la direzione. Per essere più precisi questo cambio di sistema di riferimento può essere effettuato soltanto se l'invariante relativistico $E^2 - B^2$ è conservato, per cui $\|\mathbf{E}\| < \|\mathbf{B}\|$.

Se $\|\mathbf{E}\| > \|\mathbf{B}\|$ l'invariante non è più conservato e non posso effettuare questo cambio di sistema di riferimento.

5 Particelle in prossimità di un punto a X

Posiziono $N = 10^4$ particelle in un quadrato $L \times L$ con posizione e velocità uniformemente distribuite. In particolare la velocità di ognuna sarà $|v| = 0.1$ e direzione casuale ottenuta con un angolo casuale tra 0 e 2π .

Le particelle sono immerse in un campo elettrico e magnetico:

$$\vec{E} = \left(0, 0, \frac{1}{2}\right) \quad (36)$$

$$\vec{B} = \left(\frac{y}{L}, \frac{x}{L}, 0\right) \quad (37)$$

Scrivo le equazioni del moto:

$$\begin{cases} \frac{dv_x}{dt} = 0 \\ \frac{dv_y}{dt} = 0 \\ \frac{dv_z}{dt} = \frac{1}{2} + \frac{x}{L}v_x - \frac{y}{L}v_y \end{cases} \quad (38)$$

Le particelle compiono un moto peculiare, infatti le particelle che si trovano nella zona centrale del quadrato, vengono accelerate maggiormente verso z dando effetto a una sorta di getto di particelle. Verifichiamo per quali particelle nel nostro dominio é possibile cambiare sistema di riferimento:

$$\|\mathbf{E}\| < \|\mathbf{B}\| \quad (39)$$

$$\frac{1}{2} < \sqrt{\frac{x^2}{L^2} + \frac{y^2}{L^2}} \quad (40)$$

$$x^2 + y^2 > \frac{L^2}{4} \quad (41)$$

per cui per le particelle esterne alla circonferenza di raggio $R = \frac{L}{2}$ é possibile annullare il campo \mathbf{E} e non subiranno accelerazione. Mentre le particelle interne subiranno un accelerazione verso z .

6 Conclusioni

Abbiamo considerato tre diversi esempi di particelle in campi elettrici e magnetici, in particolare per l'ultimo caso la simulazione numerica ci ha permesso di comprendere la dinamica non banale del sistema.

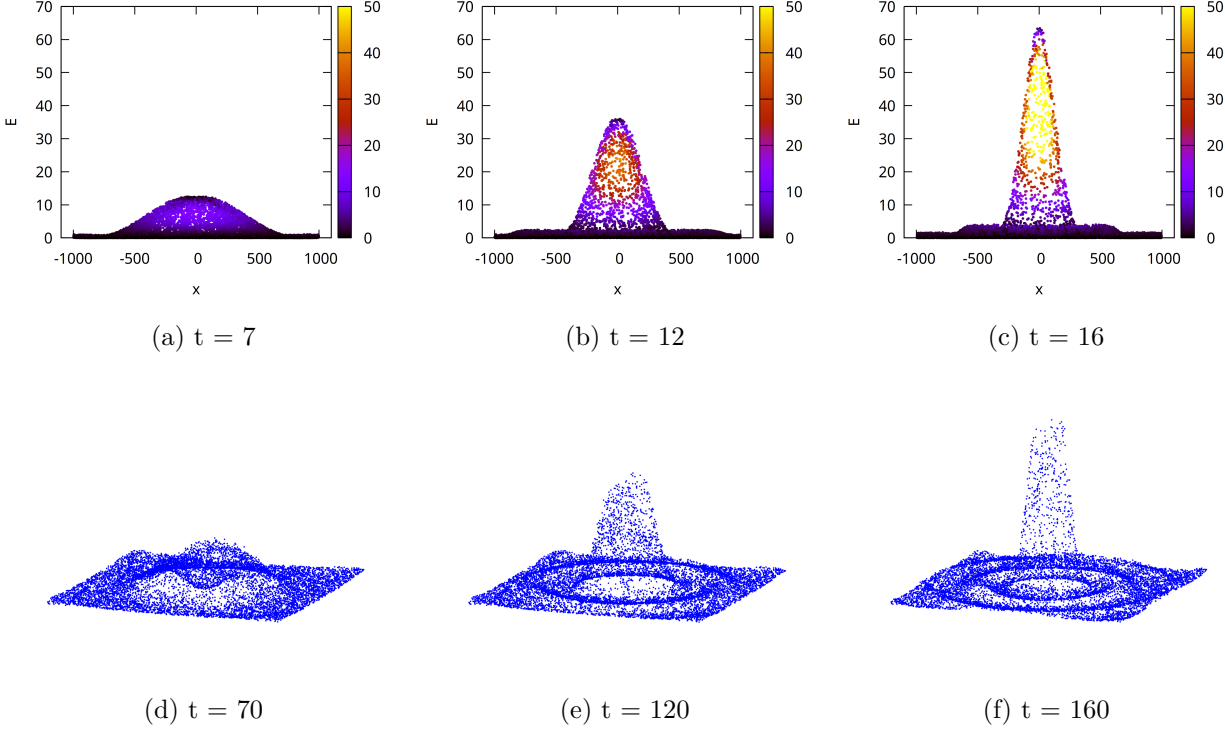


Figure 5: Posizioni delle particelle e Energia Cinetica in tre diversi istanti

Appendices

.1 Boris Pusher Algorithm

Uno step di integrazione da x^n e v^n a x^{n+1} e v^{n+1} :

$$\mathbf{x}^{n+\frac{1}{2}} = x^n + \frac{1}{2}\Delta_t \mathbf{v}^n \quad (drift)$$

$$\mathbf{v}^- = \mathbf{v}^n + \frac{1}{2}\mathbf{E}^{n+\frac{1}{2}} \quad (kick)$$

$$\mathbf{v}^+ = \mathbf{v}^- + 2 \frac{\mathbf{v}^- + \mathbf{v}^- \times \mathbf{b}}{1 + \mathbf{b}^2} \times \mathbf{b} \quad (rotate)$$

$$\mathbf{v}^{n+1} = \mathbf{v}^+ + \frac{1}{2}\mathbf{E}^{n+\frac{1}{2}} \quad (kick)$$

$$\mathbf{x}^{n+1} = \mathbf{x}^{n+\frac{1}{2}} + \frac{1}{2}\Delta_t \mathbf{v}^{n+1} \quad (drift)$$

.2 Implementazione singola particella

```
#include "prototypes.h"
```

```

void RHSFunc(double t, double * Y, double * R);
void crossproduct6(double * A, double * B, double * C);
void crossproduct3(double * A, double * B, double * C);
void Efield(double x, double y, double z, double * E);
void Bfield(double x, double y, double z, double * B);
void BorisPusher(double * Y, double dt);

#define e 1.6e-19 // qua inserisco la carica che sto considerando
#define m 9.e-31 // qua inserisco la massa che sto considerando
#define E0 1.e-2 // valore campo elettrico
#define B0 1.e-8 // valore campo magnetico

#define V0 E0 / B0
#define t0 m * V0 / (e * E0)
#define l0 V0 * t0

int main() {
    using namespace std;
    ofstream fdata;
    fdata.open("emfields.dat");
    fdata << setiosflags(ios::scientific);
    fdata << setprecision(12);
    cout << setprecision(5);

    // queste sono le velocità, le tratto come se fossero 3 semplici ODE
    int neq = 6;
    double Y[neq];

    // Condizioni Iniziali
    double x0 = 0.;
    double y0 = 0.;
    double z0 = 0.;
    double vx0 = 1.;
    double vy0 = 0.;
    double vz0 = 0.;
    double U0 = vx0 * vx0 + vy0 * vy0 + vz0 * vz0;
    double U;

```

```

// Assegnazione condizioni iniziali
Y[0] = x0; // x
Y[1] = y0; // y
Y[2] = z0; //z
Y[3] = vx0; // vx
Y[4] = vy0; // vy
Y[5] = vz0; //vz

// starting point dell' integratore
double tb = 30.;
double ta = 0.;
double t = ta;
double dt = 0.25;
int nstep = round((tb - ta) / double(dt));
double err;
// ciclo per RK2Step
for (int i = 0; i <= nstep; i++) {
    RK2Step(t, Y, RHSFunc, dt, neq);
    U = Y[3] * Y[3] + Y[4] * Y[4] + Y[5] * Y[5];
    err = fabs(U - U0) / U0;
    fdata << t << " " << Y[0] << " " << Y[1] << " " << Y[2] << " " << Y[3] << "
    t += dt;

}
fdata << endl << endl;

// ciclo per RK4Step
Y[0] = x0; // x
Y[1] = y0; // y
Y[2] = z0; //z
Y[3] = vx0; // vx
Y[4] = vy0; // vy
Y[5] = vz0; //vz
t = ta;
for (int i = 0; i <= nstep; i++) {
    RK4Step(t, Y, RHSFunc, dt, neq);
    U = Y[3] * Y[3] + Y[4] * Y[4] + Y[5] * Y[5];
    err = fabs(U - U0) / U0;

```

```

    fdata << t << " " << Y[0] << " " << Y[1] << " " << Y[2] << " " << Y[3] << " " << Y[4] << " " << Y[5] << " " << t;
    t += dt;

}

fdata << endl << endl;
// ciclo per Boris
Y[0] = x0; // x
Y[1] = y0; // y
Y[2] = z0; // z
Y[3] = vx0; // vx
Y[4] = vy0; // vy
Y[5] = vz0; // vz
t = ta;

for (int i = 0; i < nstep; i++) {
    BorisPusher(Y, dt);
    U = Y[3] * Y[3] + Y[4] * Y[4] + Y[5] * Y[5];
    err = fabs(U - U0) / U0;
    fdata << t << " " << Y[0] << " " << Y[1] << " " << Y[2] << " " << Y[3] << " " << Y[4] << " " << Y[5] << " " << t;

    t += dt;

}

fdata.close();
cout << "l0 : " << l0 << " meters" << endl;
cout << "t0 : " << t0 << " seconds" << endl;
cout << "raggio dell'orbita: " << (m * V0) / (e * B0) << "meters" << endl;
cout << "periodo dell'orbita: " << (2 * M_PI * m) / (e * B0) << "seconds" << endl;

return 0;
}

void RHSFunc(double t, double * Y, double * R) {
    double vxb[3];
    double E[3];
    double B[3];
    Efield(Y[0], Y[1], Y[2], E);

```

```

    Bfield(Y[0], Y[1], Y[2], B);
    crossproduct6(Y, B, vxb);
    //    posizioni
    R[0] = Y[3];
    R[1] = Y[4];
    R[2] = Y[5];
    //    velocita
    R[3] = E[0] + vxb[0];
    R[4] = E[1] + vxb[1];
    R[5] = E[2] + vxb[2];
}

void crossproduct6(double * A, double * B, double * C) {
    C[0] = A[4] * B[2] - A[5] * B[1];
    C[1] = -A[3] * B[2] + A[5] * B[0];
    C[2] = A[3] * B[1] - A[4] * B[0];
}

void crossproduct3(double * A, double * B, double * C) {
    C[0] = A[1] * B[2] - A[2] * B[1];
    C[1] = -A[0] * B[2] + A[2] * B[0];
    C[2] = A[0] * B[1] - A[1] * B[0];
}

void BorisPusher(double * Y, double dt) {

    double E[3];
    double B[3];
    Efield(Y[0], Y[1], Y[2], E);
    Bfield(Y[0], Y[1], Y[2], B);
    double temp[3];
    double alpha = 1;
    double upm[3], upp[3];
    double h = alpha * dt;
    double b[3];
    b[0] = 0.5 * h * B[0];
    b[1] = 0.5 * h * B[1];
    b[2] = 0.5 * h * B[2];
    double bquadro = b[0] * b[0] + b[1] * b[1] + b[2] * b[2];

```

```

double temp2[3];

Y[0] += dt * 0.5 * Y[3];
Y[1] += dt * 0.5 * Y[4];
Y[2] += dt * 0.5 * Y[5];

upm[0] = Y[3] + 0.5 * h * E[0];
upm[1] = Y[4] + 0.5 * h * E[1];
upm[2] = Y[5] + 0.5 * h * E[2];

crossproduct3(upm, b, temp);

temp2[0] = 2 * (upm[0] + temp[0]) / (1 + bquadro);
temp2[1] = 2 * (upm[1] + temp[1]) / (1 + bquadro);
temp2[2] = 2 * (upm[2] + temp[2]) / (1 + bquadro);

crossproduct3(temp2, b, temp);

upp[0] = upm[0] + temp[0];
upp[1] = upm[1] + temp[1];
upp[2] = upm[2] + temp[2];

Y[3] = upp[0] + 0.5 * h * E[0];
Y[4] = upp[1] + 0.5 * h * E[1];
Y[5] = upp[2] + 0.5 * h * E[2];

Y[0] += 0.5 * dt * Y[3];
Y[1] += 0.5 * dt * Y[4];
Y[2] += 0.5 * dt * Y[5];
}

void Efield(double x, double y, double z, double * E) {
    E[0] = 0.;
    E[1] = 0.;
    E[2] = 0.;
}

void Bfield(double x, double y, double z, double * B) {

```

```

    B[0] = 0.;
    B[1] = 0.;
    B[2] = 1.;
}

```

.3 Implementazione ExB Drift

```

#include "prototypes.h"

void RHSFunc(double t, double * Y, double * R);
void crossproduct6(double * A, double * B, double * C);
void crossproduct3(double * A, double * B, double * C);
void Efield(double x, double y, double z, double * E);
void Bfield(double x, double y, double z, double * B);
void BorisPusher(double * Y, double dt);

#define e 1.6e-19 // qua inserisco la carica che sto considerando
#define m 9.1e-31 // qua inserisco la massa che sto considerando
#define E0 1.e-2 // valore campo elettrico
#define B0 1.e-8 // valore campo magnetico

#define V0 E0 / B0
#define t0 m * V0 / (e * E0)
#define l0 V0 * t0

int main() {
    using namespace std;
    ofstream fdata;
    fdata.open("emfields.dat");
    fdata << setiosflags(ios::scientific);
    fdata << setprecision(12);
    cout << setiosflags(ios::scientific);
    cout << setprecision(12);

    // queste sono le velocità, le tratto come se fossero 3 semplici ODE
    int neq = 6;
    double Y[neq];
    // Condizioni Iniziali

```



```

double x0 = 0.;
double y0 = 0.;
double z0 = 0.;
double vx0 = 1.;
double vy0 = 0.;
double vz0 = 0.;
double U0 = vx0 * vx0 + vy0 * vy0 + vz0 * vz0;
double U;

// Assegnazione condizioni iniziali
Y[0] = x0; // x
Y[1] = y0; // y
Y[2] = z0; //z
Y[3] = vx0; // vx
Y[4] = vy0; // vy
Y[5] = vz0; //vz

// starting point del integratore
double tb = 50.;
double ta = 0.;
double t = ta;
double dt = 0.01;
int nstep = round((tb - ta) / double(dt));
double err;

Y[0] = x0; // x
Y[1] = y0; // y
Y[2] = z0; //z
Y[3] = vx0; // vx
Y[4] = vy0; // vy
Y[5] = vz0; //vz
t = ta;

for (int i = 0; i < nstep; i++) {
    BorisPusher(Y, dt);
    U = Y[3] * Y[3] + Y[4] * Y[4] + Y[5] * Y[5];
    err = fabs(U - U0) / U0;
    fdata << t << " " << Y[0] << " " << Y[1] << " " << Y[2] << " " << Y[3] << "

```

```

        t += dt;

    }

    fdata.close();
    cout << "well Done" << endl;

    return 0;
}

void RHSFunc(double t, double * Y, double * R) {
    double vxb[3];
    double E[3];
    double B[3];
    Efield(Y[0], Y[1], Y[2], E);
    Bfield(Y[0], Y[1], Y[2], B);
    crossproduct6(Y, B, vxb);
    //    posizioni
    R[0] = Y[3];
    R[1] = Y[4];
    R[2] = Y[5];
    //    velocita
    R[3] = E[0] + vxb[0];
    R[4] = E[1] + vxb[1];
    R[5] = E[2] + vxb[2];
}

void crossproduct6(double * A, double * B, double * C) {
    C[0] = A[4] * B[2] - A[5] * B[1];
    C[1] = -A[3] * B[2] + A[5] * B[0];
    C[2] = A[3] * B[1] - A[4] * B[0];
}

void crossproduct3(double * A, double * B, double * C) {
    C[0] = A[1] * B[2] - A[2] * B[1];
    C[1] = -A[0] * B[2] + A[2] * B[0];
    C[2] = A[0] * B[1] - A[1] * B[0];
}

```

```

void BorisPusher(double * Y, double dt) {

    double E[3];
    double B[3];
    Efield(Y[0], Y[1], Y[2], E);
    Bfield(Y[0], Y[1], Y[2], B);
    double temp[3];
    double alpha = 1;
    double upm[3], upp[3];
    double h = alpha * dt;
    double b[3];
    b[0] = 0.5 * h * B[0];
    b[1] = 0.5 * h * B[1];
    b[2] = 0.5 * h * B[2];
    double bquadro = b[0] * b[0] + b[1] * b[1] + b[2] * b[2];
    double temp2[3];

    Y[0] += dt * 0.5 * Y[3];
    Y[1] += dt * 0.5 * Y[4];
    Y[2] += dt * 0.5 * Y[5];

    upm[0] = Y[3] + 0.5 * h * E[0];
    upm[1] = Y[4] + 0.5 * h * E[1];
    upm[2] = Y[5] + 0.5 * h * E[2];

    crossproduct3(upm, b, temp);

    temp2[0] = 2 * (upm[0] + temp[0]) / (1 + bquadro);
    temp2[1] = 2 * (upm[1] + temp[1]) / (1 + bquadro);
    temp2[2] = 2 * (upm[2] + temp[2]) / (1 + bquadro);

    crossproduct3(temp2, b, temp);

    upp[0] = upm[0] + temp[0];
    upp[1] = upm[1] + temp[1];
    upp[2] = upm[2] + temp[2];

    Y[3] = upp[0] + 0.5 * h * E[0];

```

```

Y[4] = upp[1] + 0.5 * h * E[1];
Y[5] = upp[2] + 0.5 * h * E[2];

Y[0] += 0.5 * dt * Y[3];
Y[1] += 0.5 * dt * Y[4];
Y[2] += 0.5 * dt * Y[5];
}

void Efield(double x, double y, double z, double * E) {
    E[0] = 0.;
    E[1] = 0.5 ;
    //      E[1]=0.2; produce un plot elicoidale
    //      E[1]=1.5; produce un plot in cui la particella accelera
    E[2] = 0.;
}

void Bfield(double x, double y, double z, double * B) {
    B[0] = 0.;
    B[1] = 0.;
    B[2] = 1.;
}

```

.4 Implementazione 10k particelle

```

#include<iostream>

#include <fstream>

#include<cmath>

#include<iomanip>

const int NPART = 10000;
const double L = 1000;

#define q 1.6e-19
#define m 9.e-31

```

```

#define E0 1.e-2
#define B0 1.e-8

#define V0 E0 / B0
#define t0 m * V0 / (q * E0)
#define l0 V0 * t0

using namespace std;
void crossproduct(double * A, double * B, double * C);
void BorisPusherParticles(double x[][3], double v[][3], double dt, int NPART);
void Efield(double x, double y, double z, double * E);
void Bfield(double x, double y, double z, double * B);

int main() {
    using namespace std;
    ofstream fdata;
    fdata.open("data.dat");
    fdata << setiosflags(ios::scientific);
    fdata << setprecision(12);
    cout << setprecision(5);
    srand48(time(NULL));

    double x[NPART][3];
    double v[NPART][3];

    // starting point del integratore
    double tb = 20.;
    double ta = 0.;
    double dt = 0.1;
    int nstep = round((tb - ta) / double(dt));
    double t = ta;
    double U = 0.;

    // Assegnazione condizioni iniziali
    for (int i = 0; i < NPART; i++) {
        x[i][0] = L * (2. * drand48() - 1.); // x
        x[i][1] = L * (2. * drand48() - 1.); // y
        x[i][2] = 0.; // z
    }
}

```

```

    v[i][0] = sqrt(0.05) * cos(2 * M_PI * drand48()); // vx
    v[i][1] = sqrt(0.05) * sin(2 * M_PI * drand48()); // vy
    v[i][2] = 0.; // vz
}

// ciclo per Boris
for (int j = 0; j < nstep; j++) {
    for (int i = 0; i < NPART; i++) {
        U = v[i][0] * v[i][0] + v[i][1] * v[i][1] + v[i][1] * v[i][1];
        fdata << i << " " << t << " " << x[i][0] << " " << x[i][1] << " " << x[i]
    }
    BorisPusherParticles(x, v, dt, NPART);
    t += dt;
    fdata << endl << endl;
    if (j % 10 == 0) cout << int(t) << endl;
}
fdata.close();

return 0;
}

void crossproduct(double * A, double * B, double * C) {
    C[0] = A[1] * B[2] - A[2] * B[1];
    C[1] = -A[0] * B[2] + A[2] * B[0];
    C[2] = A[0] * B[1] - A[1] * B[0];
}

void BorisPusherParticles(double x[][3], double v[][3], double dt, int NPART) {

    double E[3] = {
        0,
        0,
        0
    };
    double B[3] = {
        0,
        0,

```

```

0
};
double temp[3];
double alpha = 1.;
double upm[3], upp[3];
double h = alpha * dt;
double b[3];
double temp2[3];
double bquadro;

for (int i = 0; i < NPART; i++) {
    Efield(x[i][0], x[i][1], x[i][2], E);
    Bfield(x[i][0], x[i][1], x[i][2], B);

    b[0] = 0.5 * h * B[0];
    b[1] = 0.5 * h * B[1];
    b[2] = 0.5 * h * B[2];
    bquadro = b[0] * b[0] + b[1] * b[1] + b[2] * b[2];

    x[i][0] += dt * 0.5 * v[i][0];
    x[i][1] += dt * 0.5 * v[i][1];
    x[i][2] += dt * 0.5 * v[i][2];

    upm[0] = v[i][0] + 0.5 * h * E[0];
    upm[1] = v[i][1] + 0.5 * h * E[1];
    upm[2] = v[i][2] + 0.5 * h * E[2];

    crossproduct(upm, b, temp);

    temp2[0] = 2 * (upm[0] + temp[0]) / (1 + bquadro);
    temp2[1] = 2 * (upm[1] + temp[1]) / (1 + bquadro);
    temp2[2] = 2 * (upm[2] + temp[2]) / (1 + bquadro);

    crossproduct(temp2, b, temp);

    upp[0] = upm[0] + temp[0];
    upp[1] = upm[1] + temp[1];
    upp[2] = upm[2] + temp[2];
}

```

```

    v[i][0] = upp[0] + 0.5 * h * E[0];
    v[i][1] = upp[1] + 0.5 * h * E[1];
    v[i][2] = upp[2] + 0.5 * h * E[2];

    x[i][0] += 0.5 * dt * v[i][0];
    x[i][1] += 0.5 * dt * v[i][1];
    x[i][2] += 0.5 * dt * v[i][2];
}
}
void Efield(double x, double y, double z, double * E) {
    E[0] = 0;
    E[1] = 0;
    E[2] = 0.5;
}
void Bfield(double x, double y, double z, double * B) {
    B[0] = y / L;
    B[1] = x / L;
    B[2] = 0;
}

```