Moto di Particelle in Campo Elettrico e Magnetico

Pietro Sillano

January 28, 2022

Indice

1 Introduzione

In numerosi campi della fisica é importante studiare la dinamica delle singole particelle, sono peró pochi i casi in cui una soluzione analitica é possibile.
Per esempio, se considero una particella carica in movimento interagisce con il campo elettromagnetico, l'equazione che ne descrive la dinamica é la seguente :

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

La soluzione analitica é 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 é la riscrittura di (1) da unitá SI in forma adimensionale, in modo da evitare eventuali cancellazioni numeriche dovute alle differenti scale delle grandezze presenti.

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

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

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

Sostituisco nell'equazione (1):

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

$$\frac{d\tilde{\mathbf{v}}}{d\tilde{t}} = \frac{qt_0E_0}{mv_0}\mathbf{E} + \frac{qt_0}{mv_0}v_0B_0 v \times \mathbf{B}$$

$$\frac{qt_0E_0}{mv_0} = 1 \qquad t_0 = \frac{mv_0}{qE_0}$$

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

$$\frac{qB_0}{m}\frac{mv_0}{qE_0} = \frac{B_0}{E_0}v_0 = 1 \qquad E_0 = B_0v_0$$

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

$$\frac{d\mathbf{v}}{dt} = \mathbf{E} + \mathbf{v} \times \mathbf{B}$$

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.

$$t_0 = \frac{mv_0}{qE_0}\ [s] \qquad v_0 = \frac{E_0}{B_0}\ \left[\frac{m}{s}\right] \qquad l_0 = v_0\ t_0\ [m]$$

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 é

hamiltoniano un integratore simplettico che conserva l'energia totale del sistema.
Gli integratori simplettici *Position-Verlet* e *Velocity-Verlet* spesso usati per integrare le equazioni del moto non sono adatti in quanto una delle ipotesi di questi metodi é che la forza sia indipendente dalla velocitá.
Ho usato quindi il *Boris Pusher* un integratore simplettico 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 analizzeró 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 velocitá della particella é parallela a B e in cui la velocitá é perpendicolare a B.

3.1 Velocitá parallela

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

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

Otteniamo una dinamica di questo tipo:

$$\begin{cases} \frac{dv_x}{dt} = 0 \\ \frac{dv_y}{dt} = 0 \\ \frac{dv_z}{dt} = 0 \end{cases}$$

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

3.2 Velocitá perpendicolare

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

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

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

Eguaglio la forza di Lorentz con la forza centripeta:

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

Calcolo ora il periodo di rotazione sostituendo R:

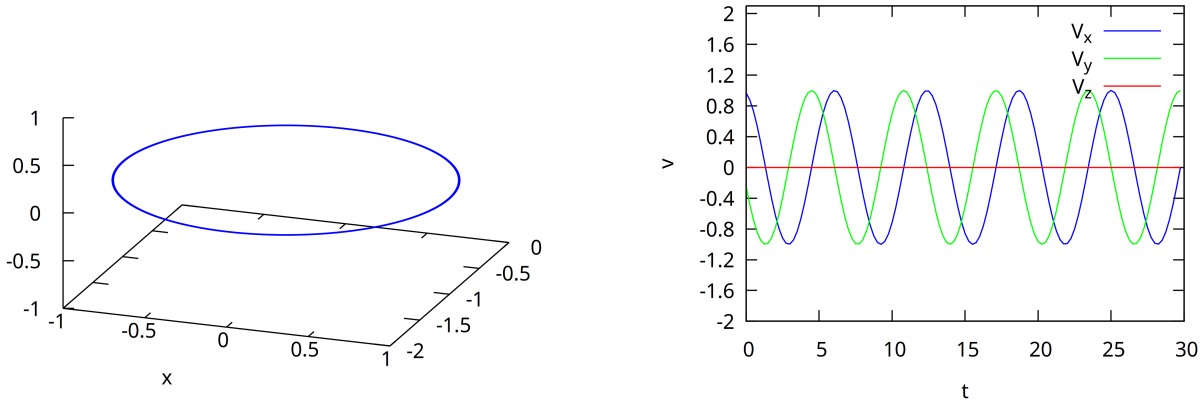
$$T = \frac{2\pi}{\omega} = \frac{2\pi R}{v} \qquad T = \frac{2\pi}{v}\frac{v}{B} = \frac{2\pi}{B} = 2\pi$$

I risultati numerici coincidono con i risultati analitici attesi se si riportano le unitá 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:

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

Per esempio volessimo convertire da grandezza adimensionale a unitá 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

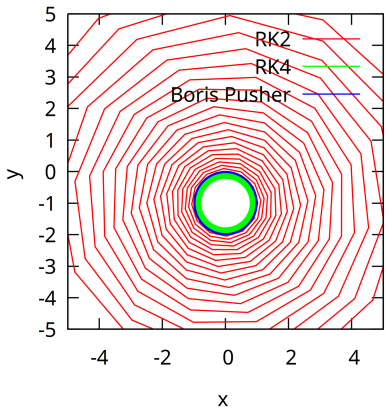
(b) Velocitá particella

Figure 1: Velocitá perpendicolare a **B**

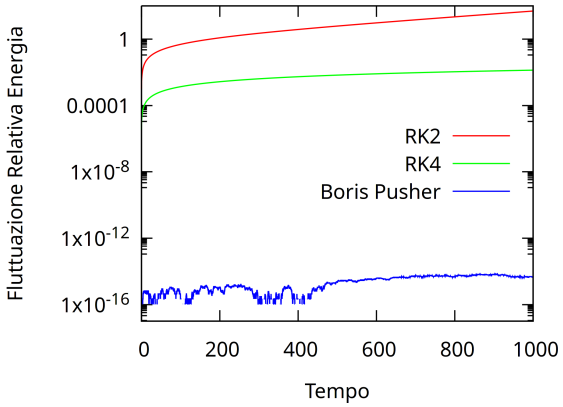
3.3 Confronto Rk2, Rk4 e Boris Pusher

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.

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



(a) Confronto tra le traiettorie



(b) Fluttuazione relativa dell'energia totale

Figure 2: Confronto tra RK2, RK4 e Boris Pusher con un timestep dt = 0.25 e 1000 step

Valuto l'errore relativo sull'energia considerando soltanto l'energia cinetica della particella in quanto il campo magnetico B non fornisce energia in quanto é sempre perpendicolare alla direzione di spostamento, influisce quindi soltanto sulla direzione della velocitá 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 a 10^{-16} e 10^{-14}. Verifichiamo che l'uso di un integratore simplettico permette la conservazione dell'energia iniziale. Nelle simulazioni seguenti sará implementato l'integratore di Boris.

4 ExB Drift

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

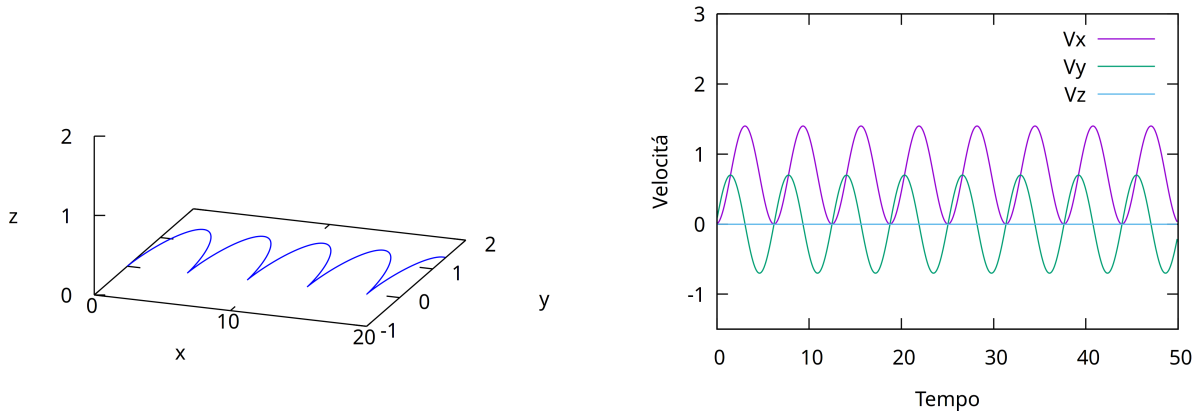
$$\mathbf{B} = (0, 0, B_z) \qquad B_z = 1$$

$$\mathbf{E} = (0, E_y, 0) \qquad E_y = 0.5$$

$$\begin{cases} \frac{dv_x}{dt} = v_y B_z \\ \frac{dv_y}{dt} = E_y \\ \frac{dv_z}{dt} = 0 \end{cases}$$

La traiettoria della particella segue una cicloide lungo la direzione x. Il moto é inaspettato avendo un campo elettrico agente in direzione y che accellera costantemente la particella lungo questa direzione.

In realtá, quando la velocitá della particella aumenta cresce anche la forza risultante dall'interazione con **B**, sempre normale rispetto alla traiettoria. Infatti la carica ferma nell'origine subisce un accelerazione lungo y dovuta a **E**. Appena la carica inizia a muoversi la forza magnetica agisce nella



(a) Traiettoria nello spazio 3d

(b) Velocitá

Figure 3: ExB Drift

direzione normale alla traiettoria. Questo fino alla fine del periodo dove la forza risultante é nulla, per cui la carica si ferma e il moto si ripete periodico.

5 10k Particelle

Posiziono $N = 10^4$ particelle in un quadrato $L \times L$ con posizione e velocitá uniformemente distribuite. In particolare la velocitá di ognuna sará $|v| = 0.1$ e direzione casuale ottenuta con un angolo casuale tra 0 e 2π.
Le particelle sono immerse in un campo elettrico e magnetico:

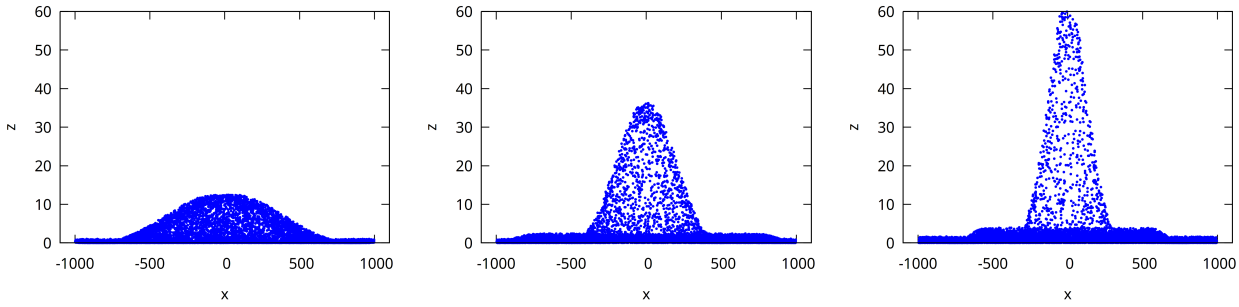
$$\vec{E} = \left(0, 0, \frac{1}{2}\right)$$

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

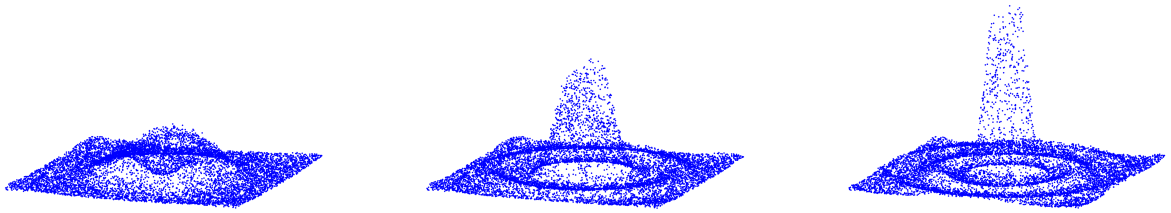
Scrivo le equazioni del moto:

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

Le particelle compiono un moto peculiare, infatti le particelle che si trovano nella zona centrale del quadrato, vengono accellerate magiormente verso z dando effetto a una sorta di getto di particelle. Mentre alla base del cono si possono osservare delle oscillazioni concentriche.



(a) t = 70 (b) t = 120 (c) t = 160



(d) t = 70 (e) t = 120 (f) t = 160

Figure 4: Posizioni delle particelle in tre diversi istanti

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.

Appendices

.1 Boris Pusher Algorithm

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

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

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

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

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

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

.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 velocitá, 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] << '
    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] << '
```

```
    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.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 << setiosflags(ios::scientific);
  cout << setprecision(12);

  //   queste sono le velocitá, 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 = 0.;
  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 = 0.1;
  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] = 0.7 ;
  //     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 = 1000;
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 = 50.;
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;
}
```