Particelle in un Campo Elettromagnetico

Kamil Laurent

August 1, 2023

Abstract

Questa relazione ha lo scopo di studiare l'interazione tra le particelle cariche e il campo elettromagnetico. Partiremo dall'equazione del moto per una particella non relativistica in un campo elettromagnetico e studieremo tre casi. Il primo caso studiato è una particella che si muove in un campo magnetico. Studieremo la traiettoria della prima particella usando due metodi di integrazione numerica: RK4 e Boris. Conoscendo la soluzione analitica del problema, lo useremo per validare gli algoritmi utilizzati anche per i passi successivi.
Il secondo caso studiato è una particella che si muove in un campo elettrico e magnetico costante. Studieremo questo caso con il metodo di Runge-Kutta al IV ordine e faremo alcune considerazioni sulla velocità della particella in vari punti.
Il terzo caso è quello di 1000 particelle in un campo elettrico costante e un campo magnetico variabile. Per questo useremo il metodo di integrazione di Boris, in modo da ridurre il numero di calcoli ad ogni iterazione. Vedremo che in questo caso non è possibile trarre conclusioni significative sul problema se non si considerano gli effetti relativistici sulla particella.

1 Definizione del Problema

In questo progetto si cercano le soluzioni alle equazioni del moto per una o più particelle in moto all'interno di un campo elettromagnetico.

L'equazione del moto è data dall'interazione della particella con il campo elettrico e il campo magnetico ed è la seguente:

$$\frac{d\vec{v}}{dt} = \frac{q}{m}(\vec{E} + \frac{\vec{v}}{c} \wedge \vec{B}) \tag{1}$$

Dove t è il tempo, $\vec{v}$ è la velocità della particella, m e q la sua massa e la sua carica ed $\vec{E}$ e $\vec{B}$ il campo elettrico e magnetico con cui la particella interagisce. Per trovare la traiettoria della particella rispetto al tempo aggiungiamo altre 3 equazioni differenziali:

$$\frac{d\vec{x}}{dt} = \vec{v} \tag{2}$$

$\vec{x}$ è la posizione della particella.

Abbiamo così un set di 6 equazioni differenziali lineari del primo ordine, che una volta risolte forniscono la traiettoria della particella nello spazio delle fasi (posizione e velocità della particella per ogni istante t).

Queste equazioni differenziali sono facilmente risolvibili solo in casi semplici (ad esempio nel caso di campo elettrico nullo e campo magnetico perpendicolare alla direzione del moto della particella, che vedremo in seguito). Per casi più complessi, come campo elettromagnetico variabile, oppure in casi in cui abbiamo un elevato numero di particelle, tali equazioni diventano estremamente complicate da risolvere analiticamente.

Per questo motivo ricorreremo a integratori numerici per trovare le soluzioni delle equazioni descritte sopra. In particolare utilizzeremo due differenti tecniche:

- Metodo Runge-Kutta al quarto ordine (RK4)
- Metodo Boris pusher per la forza di Lorentz (per particelle non relativistiche)

RK4 è una tecnica numerica usata per risolvere sistemi di equazioni differenziali ordinarie (ODE) che approssima la soluzione dell'equazione differenziale per uno step successivo ad ogni iterazione. Supponiamo di avere una equazione differenziale $\frac{dY}{dt} = f(t, Y)$ e di conoscere la condizione iniziale della soluzione (condizione al contorno). Questo metodo parte dalle condizioni iniziali (soluzioni dell'equazione nel punto iniziale) e calcola 4 funzioni ad ogni iterazione per ricavare la soluzione dell'equazione allo step successivo. Le funzioni calcolate ad ogni step sono:

- $k_1 = f(t, Y)$ (pendenza al punto iniziale)
- $k_2 = f(t + \frac{h}{2}, Y + \frac{h}{2}k_1)$ (pendenza nel punto intermedio usando k_1)
- $k_3 = f(t + \frac{h}{2}, Y + \frac{h}{2}k_2)$ (pendenza nel punto intermedio usando k_2)
- $k_4 = f(t + h, Y + hk_3)$ (pendenza al punto finale usando k_3)

Dove t è il tempo, Y è lo stato (soluzione dell'equazione) al tempo t, h è lo step temporale, $f(t, Y)$ è la funzione che ci dice come Y varia nel tempo. Una volta calcolate le quattro pendenze, possiamo dare un'approssimazione della soluzione (Y) allo step successivo:

$$Y(t + h) = Y + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{3}$$

Se abbiamo più di una equazione differenziale, questo procedimento viene ripetuto ad ogni step per ognuna delle equazioni presenti nel sistema. Il metodo RK4 commette un'errore ad ogni step dell'ordine di h^4, ha quindi una precisione molto elevata ma richiede un grande numero di calcoli ad ogni iterazione.

Il metodo di Boris è invece specifico per risolvere le equazioni del moto per una particella carica in un campo elettromagnetico. La sua accuratezza è di ordine h^2 ma usa meno passaggi per ricavare la soluzione delle equazioni allo step successivo, per cui è più indicato per problemi che richiedono un elevato numero di calcoli. Inoltre è migliore se si usa su tempi lunghi, perché il suo errore oscilla e non viene accumulato ad ogni step, come invece succede per RK4.

I calcoli svolti ad ogni step dall'integratore di Boris sono i seguenti:

- $\mathbf{x} = \mathbf{x} + \frac{h}{2}\mathbf{v}$ (posizione al tempo intermedio)
- $\mathbf{v}_- = \mathbf{v} + \frac{q\mathbf{E}}{2m}h$ (velocità allo step intermedio)
- $\mathbf{v}_+ = \mathbf{v}_- + 2\frac{\mathbf{v}_- + \mathbf{v}_- \times \mathbf{b}}{1+\mathbf{b}^2} \times \mathbf{b}$ (rotazione della velocità)
- $\mathbf{v} = \mathbf{v}_+ + \frac{q\mathbf{E}}{2m}h$ (velocità al tempo finale)
- $\mathbf{x} = \mathbf{x} + \frac{h}{2}\mathbf{v}$ (posizione al tempo finale)

con $\mathbf{b} = \frac{q}{2m}h\mathbf{B}$. Ad ogni iterazione l'algoritmo calcola la posizione e la velocità della particella, che sono le soluzioni al nostro set di equazioni. Si può vedere l'implementazione di entrambi gli algoritmi in appendice.

Il primo integratore è stato utilizzato nello studio della traiettoria di una singola particella per via della sua precisione migliore per tempi brevi. Il secondo metodo è stato usato nella simulazione che coinvolge 1000 particelle, per via del minor numero di passaggi svolti ad ogni iterazione.

Per procedere, l'equazione del moto è stata resa adimensionale scegliendo come costanti di riferimento:

- $m_{cgs} = m_e$ (massa dell'elettrone)
- $q_{cgs} = -e$ (carica dell'elettrone)
- $v_{cgs} = c$ (velocità della luce)
- $t_{cgs} = s$ (secondi)

Otteniamo dunque le quantità adimensionali: $\vec{v_a} = \vec{v}/c$, $t_a = t/s$, $\vec{x_a} = \vec{x} * cs$, $q = q_a/e$, $m = m_a/m_e$, $\vec{E} = \vec{E_a}/\frac{cm_e}{se}$ e $\vec{B} = \vec{B_a}/\frac{cm_e}{se}$

In queste unità "naturali", le equazioni diventano:

$$\frac{d\vec{v_a}}{dt_a} = \frac{q_a}{m_a}(\vec{E_a} + \vec{v_a} \wedge \vec{B_a}) \tag{4}$$

$$\frac{d\vec{x_a}}{dt_a} = \vec{v_a} \tag{5}$$

prendendo in considerazione un elettrone abbiamo $m_a = 1$ e $q_a = -1$. D'ora in avanti ometteremo il pedice a, e le quantità considerate saranno sempre adimensionali.

2 Validazione dei Metodi Numerici Utilizzati

Prima di procedere con casi più complicati, validiamo gli integratori numerici che utilizzeremo confrontando i loro risultati con le soluzioni analitiche note di un problema semplice. Consideriamo un elettrone non relativistico posizionato in $\vec{x}(t = 0) = (0, 0, 0)$ con velocità iniziale $\vec{v}(t = 0) = (0, 0.1, 0.1)$.

Supponiamo che l'elettrone sia immerso in un campo elettrico nullo ($\vec{E} = (0, 0, 0)$) e un campo magnetico costante $\vec{B} = (0, 0, 1)$

La traiettoria per una particella che si muove in un campo magnetico costante e perpendicolare alla direzione del moto è un cerchio caratterizzato dal raggio di Larmor. Possiamo scomporre il moto della particella considerata in due parti:

- un moto circolare uniforme di raggio $r_L = \frac{mv_\perp}{|q|B}$ sul piano perpendicolare al campo magnetico (piano (x, y))
- un moto lineare uniforme lungo l'asse parallelo alla direzione del campo magnetico (asse z)

Con le unità scelte e le condizioni iniziali che abbiamo definito, abbiamo una traiettoria sul piano (x, y) centrata nel punto $(-0.1, 0)$ e di raggio $r_L = 0.1$. Le equazioni del moto della particella sono quindi:

$$x(t) = 0.1 \cos t - 0.1 \tag{6}$$

$$y(t) = 0.1 \sin t \tag{7}$$

$$z(t) = 0.1 * t \tag{8}$$

Per studiare l'errore commesso dagli integratori numerici nel calcolo della posizione e della velocità della particella, prendiamo un intervallo temporale $t = 100$ e plottiamo l'errore commesso in funzione della grandezza degli step. L'errore ϵ è calcolato prendendo il valore assoluto della differenza tra il risultato numerico ottenuto ($soluzione(t = 100)$)e il valore noto al tempo finale.

$$\epsilon = |soluzione(t = 100) - 0.1 * \cos 100 - 0.1| \tag{9}$$

Per fare questa analisi consideriamo l'errore commesso sul calcolo della coordinata x per la traiettoria della particella. Il codice che calcola l'errore per step temporali di diverse grandezze - visibile in appendice:

- divide l'intervallo temporale in 128 parti uguali e applica l'algoritmo RK4 o Boris.
- calcola l'errore commesso al tempo finale $t = 100$
- raddoppia il numero di step e ripete il processo
- si ferma quando arriva a 16384 intervalli

in figura 1 possiamo osservare il plot dell'errore commesso in funzione del time step usato dall'integratore numerico. Gli assi sono in scala logaritmica.

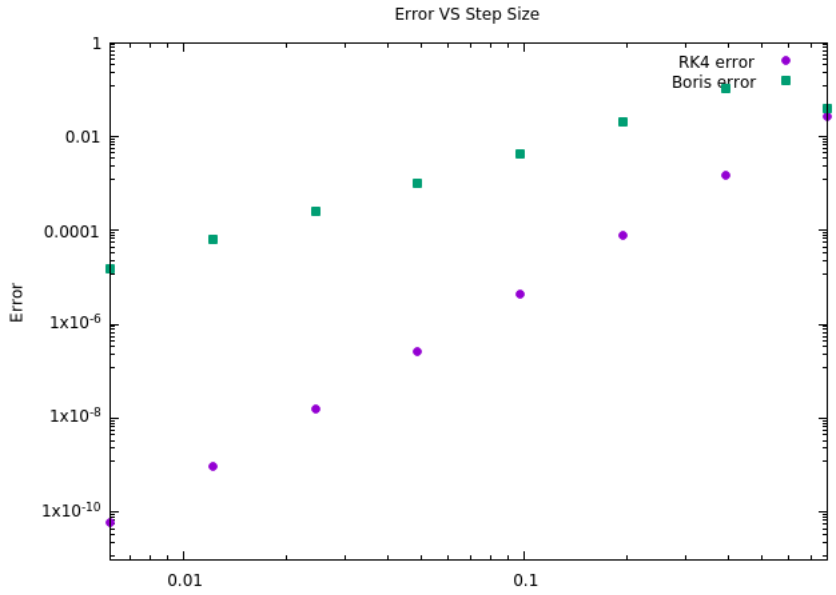


Figure 1: Errore Commesso in funzione del Time Step

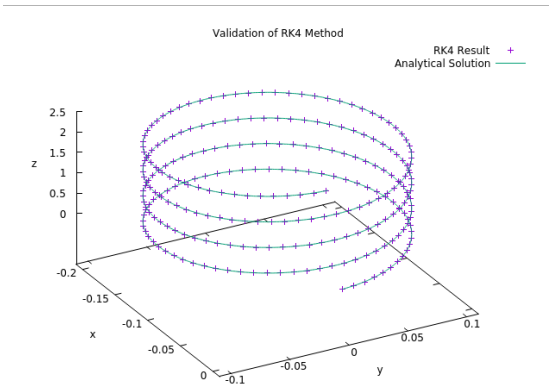
Da questo plot osserviamo come l'algoritmo RK4 sia più accurato per periodi temporali piccoli ($t = 100$ in questo caso). Il vantaggio di usare l'algoritmo di Boris è quello di svolgere meno calcoli ad ogni iterazione, per cui è più indicato nelle simulazioni che coinvolgono un gran numero di particelle.

L'algoritmo di Boris (e più in generale gli integratori simplettici) è più vantaggioso anche nel caso di simulazioni su tempi molto lunghi. Questo perché l'errore nei metodi di Runge-Kutta si propaga e continua a crescere indefinitamente, mentre negli integratori simplettici oscilla e non supera mai un certo valore. Quindi, per un grandissimo numero di step l'errore commesso da RK4 diventa paragonabile a quello commesso da Boris, e questo non giustifica più il maggior numero di passaggi fatti da RK4 ad ogni iterazione.

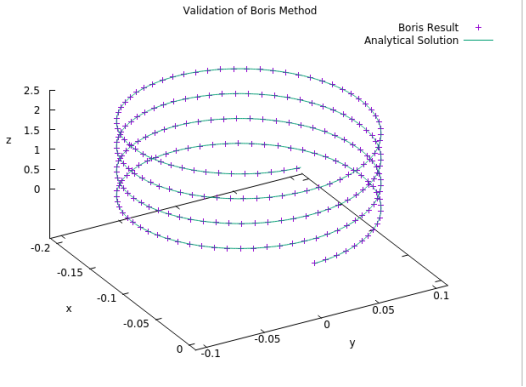
Per i prossimi casi scegliamo un time step fisso $\Delta t = 0.1$. Questo ci da una buona accuratezza sia per le traiettorie calcolate usando il metodo di Boris che per quelle ricavate con RK4.

Nelle figure 2(a) e 2(b) è rappresentata graficamente la soluzione analitica della traiettoria della particella e confrontata con le soluzioni ottenute usando i due metodi numerici. Le traiettorie combaciano anche graficamente.

Consideriamo i due metodi numerici validati e proseguiamo nella soluzione di problemi più complessi.



(a) Confronto fra Soluzione Analitica e RK4



(b) Confronto fra Soluzione Analitica e Metodo di Boris

Figure 2: Confronto fra Risultati degli Integratori Numerici e Soluzioni Analitiche

3 Particella in un Campo Elettromagnetico Costante

Studiamo ora una particella (elettrone) che si muove in un campo elettromagnetico costante: $\vec{E} = (0, 0.1, 0)$, $\vec{B} = (0, 0, 1)$.

Utilizziamo il metodo RK4 con un time step $\Delta t = 0.1$ e condizioni iniziali sulla particella $\vec{v} = (0, 0.1, 0)$ e $\vec{x} = (0, 0, 0)$. Il codice utilizzato è visibile in appendice.

Essendo la velocità iniziale diretta solo lungo l'asse y, con campo elettrico parallelo alla velocità e campo magnetico perpendicolare, ci aspettiamo che la traiettoria della particella giaccia sul piano (x, y). La traiettoria della particella ottenuta dopo un tempo $t = 20s$ è riportata nella figura 3.

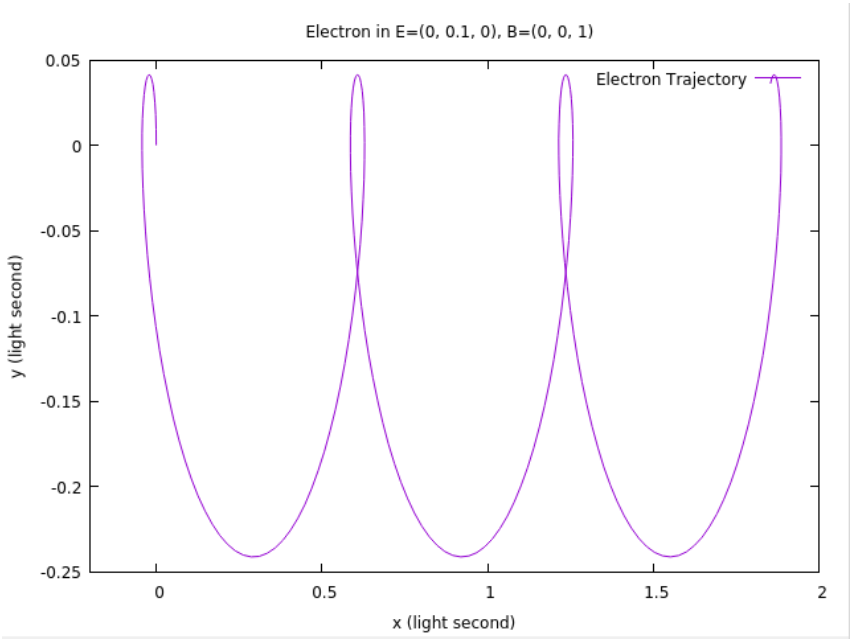


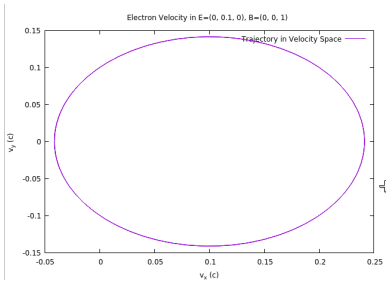
Figure 3: Traiettoria di un Elettrone in un Campo Costante

Come si può osservare in figura 3, la particella subisce l'effetto del campo magnetico (rotazione sul

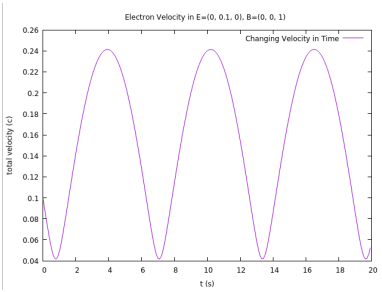
piano x, y) e del campo elettrico (accelerazione lungo l'asse y). Osserviamo un moto periodico della particella. Il campo elettrico accelera l'elettrone, e il campo magnetico cambia la sua direzione. Per cui quando l'elettrone ha una velocità concorde con la direzione di $\vec{E}$ viene rallentato, viceversa viene accelerato.

Questo è in accordo con il principio di conservazione dell'energia, perché la particella si muove in un potenziale elettrico che varia. Per fare in modo che l'energia totale della particella rimanga costante, anche l'energia cinetica (e quindi la velocità) della particella deve variare. La velocità varia in modo periodico per effetto del campo magnetico che cambia la direzione dell'elettrone.

Possiamo plottare la traiettoria della particella nello spazio delle velocità, osserviamo una traiettoria chiusa (figura 4a). Questo significa che la velocità cambia periodicamente. Il moto periodico della velocità della particella può essere visto più esplicitamente in figura 4(b) dove viene plottata la velocità totale al variare del tempo.



(a) Traiettoria dell'elettrone nello Spazio delle Velocità



(b) Velocità Totale VS Tempo

Figure 4: la Velocità Totale della Particella Cambia Periodicamente

3.1 Soluzione Analitica

Anche in questo caso possiamo ricavare una soluzione analitica da confrontare con la soluzione numerica.

In particolare consideriamo un sistema in cui $\vec{B} > \vec{E}$. Sappiamo che la quantità $\vec{B}^2 - \vec{E}^2$ è un invariante per trasformazioni di coordinate. Avendo un sistema in cui $\vec{B}^2 - \vec{E}^2 > 0$, possiamo metterci in un altro sistema di riferimento (che chiamiamo X' in cui si annulla $\vec{E}$ e rimane solo $\vec{B}$. In questo caso la soluzione analitica sarebbe quella ricavata nel caso precedente. Facciamo poi la trasformazione di coordinate inversa per ottenere la soluzione analitica nel primo sistema di riferimento (X). I due campi trasformano in questo modo cambiando sistema di riferimento inerziale:

$$\vec{E}' = \gamma(\vec{E} + \vec{v} \wedge \vec{B}) \tag{10}$$

$$\vec{B}' = \gamma(\vec{B} - \vec{v} \wedge \vec{E}) \tag{11}$$

Vogliamo annullare la componente y del campo elettrico nel nuovo sistema di riferimento (X'). Ponendo $\gamma = 1$ otteniamo:

$$E'_y = E_y + v_z B_x - v_x B_z = 0 \tag{12}$$

Se $\vec{B} = (0, 0, 1)$ e $\vec{E} = (0, E_y, 0)$, l'unico modo per annullare il campo elettrico è prendere un sistema di riferimento con velocità $\vec{v} = (E_y, 0, 0)$. In questo sistema otteniamo

$$B'_z = B_z - v_x B_y \tag{13}$$

Con le condizioni iniziali che abbiamo scelto in X, in X' abbiamo una particella che si muove con $\vec{v}'_p = (-0.1, 0.1, 0)$ in un campo elettrico nullo e un campo magnetico $\vec{B}' = (0, 0, 1.01)$. La sua traiettoria è un moto circolare uniforme con raggio $r'_L = \frac{mv_\perp}{|q|B'}$ centrata in $\vec{x}' = (r'_L/\sqrt{2}, r'_L/\sqrt{2}, 0)$. Consideriamo inoltre che a $t = 0$ la retta che congiunge il centro del cerchio e la posizione della particella è inclinata di 45 deg. Questo significa che dobbiamo aggiungere una fase di $\pi/4$ per avere la particella posizionata nell'origine a $t = 0$. Con queste considerazioni concludiamo che la traiettoria sta sul piano (x', y') con equazioni del moto

$$x'(t) = r_L \cos(t + \pi/4) - \frac{r_L}{\sqrt{2}} \tag{14}$$

$$y'(t) = r_L \sin(t + \pi/4) - \frac{r_L}{\sqrt{2}} \tag{15}$$

A questo punto torniamo nel sistema di riferimento X usando le trasformazioni di Galileo. In X le equazioni del moto diventano:

$$x(t) = r_L \cos(t + \pi/4) - \frac{r_L}{\sqrt{2}} - v_x t \tag{16}$$

$$y(t) = r_L \sin(t + \pi/4) - \frac{r_L}{\sqrt{2}} \tag{17}$$

Questa è la soluzione analitica che stavamo cercando, la possiamo confrontare graficamente con la soluzione ricavata numericamente. In figura 5 possiamo osservare la traiettoria di un elettrone dopo $t = 20s$ ottenuta analiticamente e quella ricavata con il calcolo numerico.

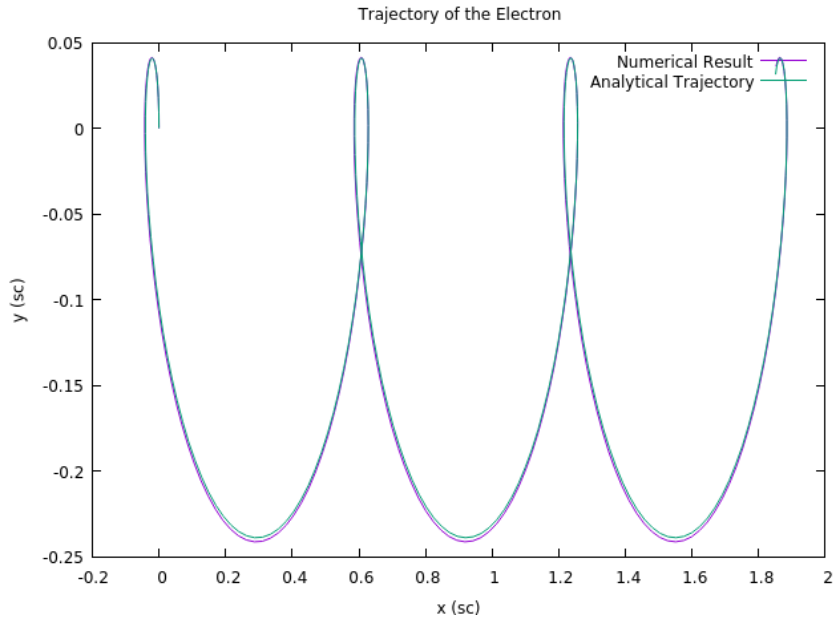


Figure 5: Confronto tra Traiettoria Analitica e Traiettoria Ottenuta Numericamente

Le due traiettorie nel grafico non combaciano perfettamente in alcune regioni. Per capire se il risultato numerico è attendibile, analizziamo l'errore commesso da RK4 in funzione del tempo. In figura 6 possiamo vedere che l'errore oscilla e per tempi brevi non supera mai il valore di 0.0025, consideriamo quindi questo errore accettabile, anche se ci aspettiamo una discrepanza minore usando un time step $\Delta t = 0.1$.

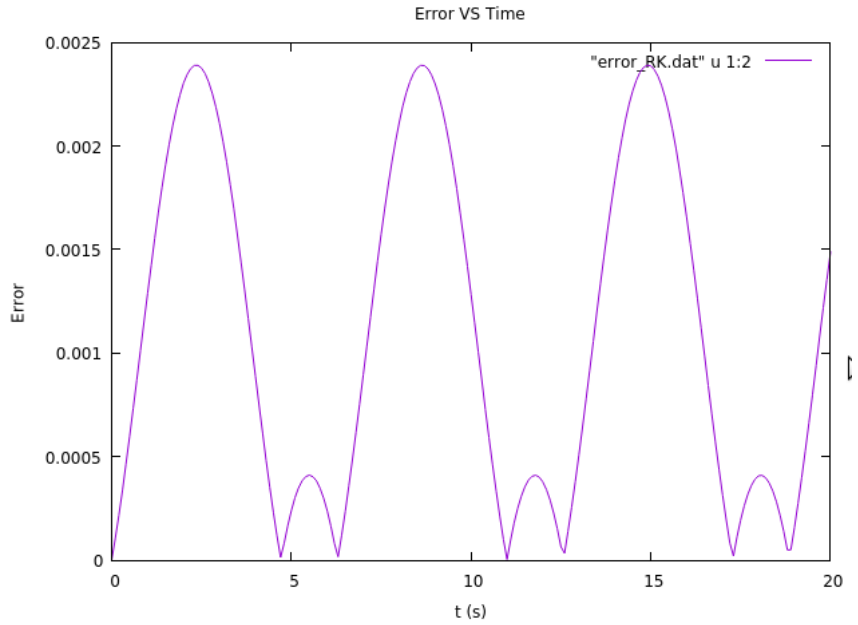


Figure 6: Errore sulla Coordinata x in Funzione del Tempo (usando un time step di 0.1)

Va sottolineato il fatto che la traiettoria analitica ricavata con un cambio di sistema di riferimento in cui $\vec{E} = 0$ non sarebbe stata possibile nel caso in cui il campo elettrico fosse stato maggiore del campo magnetico. In questo caso non avremmo potuto trovare un sistema in cui $\vec{E} = 0$. Avremmo quindi avuto una particella accelerata nel tempo in ogni sistema di riferimento. La sua velocità avrebbe avuto un moto periodico ma l'orbita nello spazio delle fasi non sarebbe stata chiusa, il che significa che non sarebbe tornata al valore di partenza ma ad ogni ciclo sarebbe aumentata.

4 1000 Particelle in un Campo Elettromagnetico Variabile

Consideriamo ora 1000 particelle distribuite casualmente su un piano $[-L, L]^2$ con velocità $|v| = 0.1$ e direzioni iniziali distribuite in modo casuale. Con queste condizioni iniziali, accendiamo un campo elettrico lungo la direzione z: $\vec{E} = (0, 0, 1/2)$, e un campo magnetico perpendicolare a $\vec{E}$ che varia in questo modo: $\vec{B} = (y/L, x/L, 0)$. Facciamo evolvere il sistema per $t = 100s$ e studiamo le traiettorie delle particelle.

Per questo sistema, dato l'elevato numero di calcoli che deve eseguire l'algoritmo, utilizziamo il metodo di Boris, che riduce il numero di funzioni che deve calcolare il programma ad ogni iterazione rispetto a RK4. In questo caso l'algoritmo di Boris è stato modificato ponendo $\gamma = 1$ in modo da non tenere conto degli effetti relativistici del sistema.

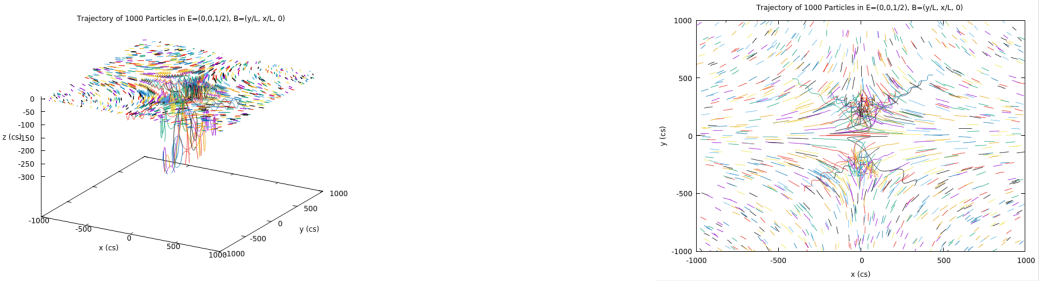
Per inizializzare il sistema usiamo un generatore di numeri casuali per ottenere le coordinate delle particelle sul piano (x, y) (generando due numeri da -1000 a 1000). Successivamente assegniamo le componenti della velocità iniziale di ogni particella generando casualmente due angoli $\theta \in (0, \pi)$ e $\phi \in (0, 2\pi)$. Definiamo le tre componenti della velocità per ciascuna particella come:

$$v_x = 0.1 \sin\theta \cos\phi \tag{18}$$
$$v_y = 0.1 \sin\theta \sin\phi \tag{19}$$
$$v_z = 0.1 \cos\theta \tag{20}$$

Visto che consideriamo particelle non interagenti tra di loro, è possibile trattare separatamente ogni singola particella. Per ottenere un plot che comprende le traiettorie spaziali di 1000 particelle:

- definiamo posizione e velocità iniziali della particella
- usiamo l'algoritmo di Boris per far evolvere la posizione e la velocità della particella nel tempo, ad ogni step scriviamo su un file .dat la posizione e la velocità della particella
- trascorso il periodo scelto, scriviamo su un secondo file l'energia cinetica della particella $E_k = \frac{1}{2}v^2$
- ripetiamo per mille volte

In figura 7.a e 7.b possiamo osservare i plot 3D e 2D di tutte le traiettorie spaziali delle 1000 particelle dopo un periodo trascorso di 100 secondi.



(a) Traiettorie delle particelle in 3 dimensioni

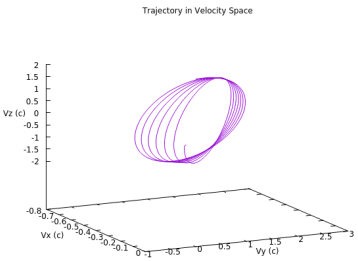
(b) Traiettorie delle Particelle sul Piano (x,y)

Figure 7: Traiettorie di 1000 Particelle dopo 100 secondi

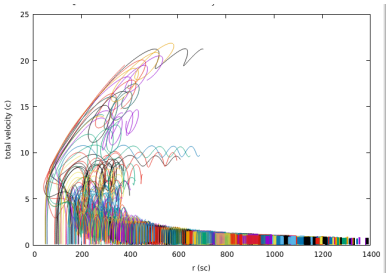
Dai due grafici è evidente che le velocità delle particelle aumentano andando verso il centro del piano, perché le traiettorie diventano più lunghe a parità di tempo trascorso. In questo caso, a differenza del caso precedente, le velocità delle particelle non oscillano tra due valori fissi, questo si può vedere plottando la

traiettoria di una delle particelle nello spazio delle velocità e osservando che non è una traiettoria chiusa (figura 8.a).

Si può vedere chiaramente che le velocità delle particelle aumentano andando verso il centro del grafico in figura 8.b, dove sono plottate le velocità totali di tutte le particelle in funzione della distanza dal centro. In questo secondo plot non è possibile distinguere la traiettoria nello spazio delle velocità della singola particella, ma l'andamento generale è un aumento della velocità verso il centro del sistema.



(a) Traiettoria della Velocità di una Test Particle



(b) Velocità Totale delle Particelle VS Distanza da Centro

Figure 8: Le particelle hanno un moto accelerato

Il fatto che le particelle accelerino sempre di più spostandosi verso il centro del volume è chiaro, consideriamo diverse zone del nostro sistema:

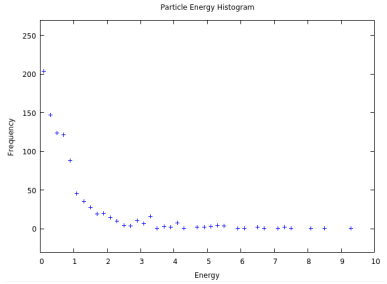
- ai bordi del volume il campo magnetico è maggiore del campo elettrico, quindi abbiamo una situazione analoga al caso precedente, in cui la particella viene accelerata da $\vec{E}$ e la sua direzione viene cambiata da $\vec{B}$. Come nel caso precedente, la velocità della particella oscilla, anche se non torna mai al valore di partenza per via del campo magnetico variabile
- andando verso il centro, il campo magnetico decresce fino al punto da diventare minore del campo elettrico. In questo caso non possiamo più annullare $\vec{E}$ in nessun sistema di riferimento, la particella accelera in ogni sistema di riferimento che consideriamo. L'accelerazione è comunque smorzata dall'effetto del campo magnetico che cambia la direzione della particella
- al centro del volume considerato, il campo magnetico diventa trascurabile rispetto al campo elettrico. La particella risente solo degli effetti di $\vec{E}$ e subisce un'accelerazione costante lungo la direzione $-z$ (da qui il fatto che le traiettorie in centro sono molto allungate rispetto a quelle ai lati

4.1 Energia del Sistema e Legge di Potenza

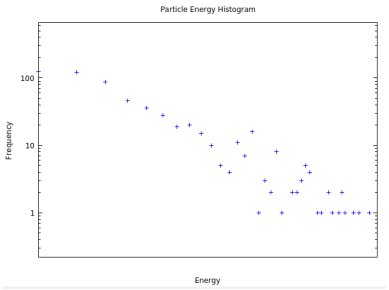
Oltre a vedere che le particelle in questo sistema accelerano con il passare del tempo, possiamo studiare la distribuzione di energia cinetica delle particelle dopo un tempo t.

In particolare eseguiamo una simulazione per un periodo di tempo $t = 100s$, poi scriviamo l'energia cinetica della particella ($E_k = 1/2v^2$) su file. Facciamo questo per ognuna delle 1000 particelle. Dopodiché usiamo Gnuplot per creare un istogramma delle energie delle particelle, dividendo lo spettro dell'energia in intervalli $\Delta E = 0.2$ e contando il numero di particelle ad ogni intervallo. Gli script eseguibili con Gnuplot possono essere consultati in appendice.

In figura 9 è rappresentato l'istogramma ottenuto, sia in scala standard che in scala logaritmica. Possiamo osservare che l'istogramma in scala logaritmica ricorda l'andamento di una retta. Questo andamento è tipico di una distrubuzione nota come "power law" o "legge di potenza"



(a) Istogramma della Distribuzione delle Energie Cinetiche



(b) Istogramma in Scala Logaritmica

Figure 9

La legge di potenza è una distribuzione che si presenta in moltissimi ambiti della fisica, e addirittura in sociologia ed economia. Questa segue una curva esponenziale decrescente:

$$y = Ax^{-b} \tag{21}$$

La legge di potenza ha le proprietà di essere un invariante di scala e di seguire una retta quando gli assi x e y sono in scala logaritmica. Il fatto di essere invariante di scala significa che in ogni scala la curva mantiene la sua proporzionalità: ad esempio, in questo caso specifico, se per ogni particella con energia $E_k = 2$ ne esistono 4 con energia $E_k = 1$, allora per ogni particella con energia $E_k = 1$ ne esistono 4 con energia $E_k = 1/2$.

Non ci resta che trovare i parametri della curva facendo un fit dei dati. Per fare questo sono stati usati due codici eseguibili da Gnuplot:

- un codice che prende i dati delle energie delle particelle, conta quante particelle ci sono per ogni intervallo di energia e stampa il punto centrale dell'intervallo e il numero di particelle di quell'intervallo in due colonne di un file .dat
- un secondo codice che legge il file generato in precedenza e genera il fit dei dati con una curva

Fittando i dati con una curva del tipo $f(x) = Ax^{-b}$, otteniamo come parametri della curva:

$$A = 54.42 \pm 2.99$$

$$b = 1.42 \pm 0.09$$

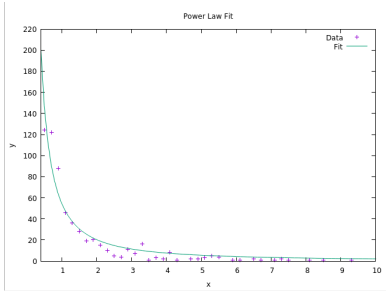
In scala logaritmica, con $x' = \log x$ e $y' = \log y$ otteniamo una retta $y' = mx' + q$ i cui parametri sono

$$m = -b$$

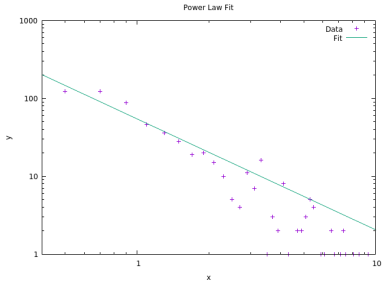
$$q = \log A$$

In figura 10 si può osservare il fit della curva con i dati ottenuti dalla simulazione, sia in scala normale che in scala logaritmica.

Da questa distribuzione possiamo estrapolare la probabilità che una particella messa in un punto casuale sul piano (x, y) con velocità $|v| = 0.1$ diretta in una direzione casuale, abbia una particolare



(a) Fit della Distribuzione delle Energie Cinetiche



(b) Fit in Scala Logaritmica

Figure 10: Fit dell'istogramma con una power law $f(x) = Ax^{-b}$

energia cinetica dopo un intervallo di tempo $t = 100s$. In particloare la probabilità che questa particella abbia un'energia E_k dopo $100s$ è

$$P(E_k) = A(E_k)^{-b} \tag{22}$$

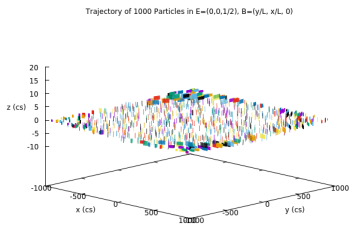
con A e b parametri ricavati in precedenza.

5 Correzioni Relativistiche

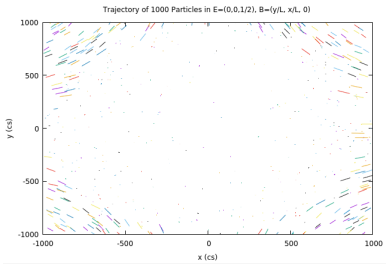
I risultati ottenuti nelle situazioni precedenti, sono frutto dell'equazione del moto non relativistica (equazione 1). Questo significa che gli effetti relativistici del problema non sono tenuti in considerazione, le particelle considerate possono accelerare all'infinito e non esiste un limite alle velocità. Questo effetto si può osservare esplicitamente nel grafico in figura 8.b, dove si vede che le velocità totali delle particelle superano la velocità della luce.

Verifichiamo che il problema sta nel fatto di aver usato equazioni non relativistiche. In questo caso, correggendo gli algoritmi usati in modo che tengano conto degli effetti relativistici, dovremmo ottenere delle traiettorie delle particelle più corte e delle velocità che non superano mai il valore di c (velocità della luce).

Modificando l'algoritmo di Boris per il caso relativistico otteniamo le traiettorie in figura 11 di 1000 particelle dopo un tempo $t = 100s$



(a) Plot 3D delle Traiettorie di 1000 Particelle Relativistiche



(b) Plot 2D delle Traiettorie di 1000 Particelle Relativistiche

Figure 11

Dai due grafici si può osservare che le traiettorie delle particelle nel centro del volume considerato sono molto più corte rispetto al caso non relativistico. Per verificare esplicitamente che il limite della velocità della luce sia rispettato, grafichiamo le velocità totali delle 1000 particelle in funzione della distanza dal centro. Osserviamo in figura 12 che il limite alla velocità totale delle particelle è la velocità della luce. Le leggi fisiche in questo caso vengono rispettate.

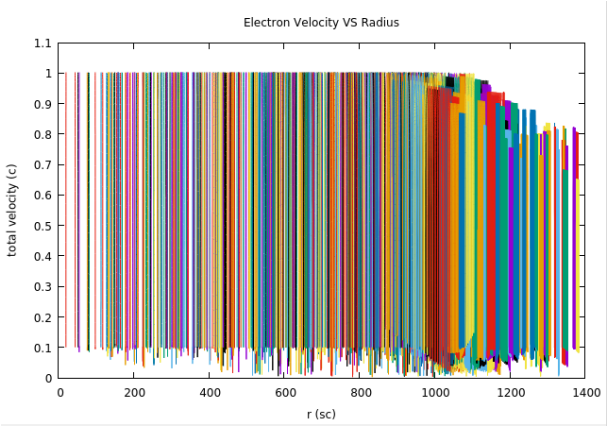


Figure 12: Velocità totale delle particelle in funzione della distanza dall'origine

A Codice Utilizzato

Qui è riportato il codice utilizzato nelle simulazioni.

```
# define SECTION 4 // Define the part of the code that you want to use

void dYdt(double, double *, double *);
void RK4(double, double *, void(double, double *, double *), double, double);
void Boris(double *, double *, double *, double *, double, int);

int main(){
    cout<< setiosflags(ios::scientific)<<setprecision(12);
    ofstream fdata;
    ofstream flog;

    // define the initial conditions and the constants
    double t0 = 0.0; // Initial time
    double tf = 100.0; // Final time
    double dt = 0.1; // Time step
    int Neq = 14; // Number of equations
    int i;
    double t;

    // Analytical formula for a known case (particle in B = (0, 0, 1))
    double x_pos, y_pos, z_pos;
    t = 0.0;
    fdata.open("analytical_trajectory.dat");
    for (i = 0; i <= 10000; i ++) {
        // Compute the current time and state to the output file
        x_pos = 0.1 * cos(t)  -0.1;
        y_pos = 0.1 * sin(t);
        z_pos = 0.1 * t;
        fdata << t << " " << x_pos << " " << y_pos << " " << z_pos << endl;
        t += 0.01;
    }
    fdata.close();

    #if SECTION == 1
    // Initial conditions
    double Y[14] = {0.0, 0.0, 0.0, 0.0, 0.1, 0.1, -1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    1.0};
    double err_x;

    // Open the output file
    fdata.open("particle_1.dat");
    // Loop to perform integration steps
    for (t = t0; t <= tf; t += dt) {
        // Print the current time and state to the output file
        fdata << t << " ";
        for (i = 0; i < Neq; i++) {
            fdata << Y[i] << " ";
        }
        fdata << endl;

        // Call the Runge-Kutta method to update the state
        RK4(t, Y, dYdt, dt, Neq);
    }
    fdata.close();


    //Open the "error" output file
    fdata.open("error_RK4.dat");
    double x_tf;

    // Compute the system's error using different time steps
    for (i = 128; i <= 16384; i *= 2){
        // Reset initial conditions
        Y[0] = Y[1] = Y[2] = Y[3] = 0.0;
```

```
        Y[4] = Y[5] = 0.1;
        t = t0;
        dt = fabs(tf - t0)/double(i); // Define a new time step

        for (t = t0; t < tf; t += dt) {
            // Call the Runge-Kutta method to update the state
            RK4(t, Y, dYdt, dt, Neq);
        }
        // Print the error at t = tf using i time steps
        x_tf = 0.1 * cos(t) - 0.1;
        fdata << dt << " " << fabs(Y[0] - x_tf) << " " << Y[0] << " " << x_tf << endl;
    }
    fdata.close();

    # elif SECTION == 2

    int N = 3;  // Number of dimensions
    double x[N] = {0.0};  // Initial position
    double v[N] = {0.0, 0.1, 0.1};  // Initial velocity
    double E[N] = {0.0, 0.0, 0.0};  // Electric field
    double B[N] = {0.0, 0.0, 1.0};  // Magnetic field

    // Check if the algorithm works with a known case
    fdata.open("particle_2.dat");
    for (t = t0; t<tf; t+=dt){

        // Print current state in the output file
        fdata << t << " " << x[0] << " " << x[1] << " " << x[2];
        fdata << " "<< v[0] << " " << v[1] << " " << v[2] << endl;

        // Call the Boris integrator
        Boris(x, v, E, B, dt, N);
    }
    fdata.close();

    //Open the "error" output file
    fdata.open("error_Boris.dat");
    double x_tf;

    // Compute the error for different time steps
    for (i = 128; i <= 16384; i *= 2){
        // Reset initial conditions
        x[0] = x[1] = x[2] = v[0] = 0.0;
        v[1] = v[2] = 0.1;
        t = t0;
        dt = fabs(tf - t0)/double(i); // Define a new time step

        for (t = t0; t < tf; t += dt) {
            // Call the Boris method to update the state
            Boris(x, v, E, B, dt, N);
        }
        // Print the error at t = tf using i time steps
        x_tf = 0.1 * cos(t) - 0.1;
        fdata << dt << " " << fabs(x[0]-x_tf) << " " << x[0] << " " << x_tf << endl;
    }
    fdata.close();

    #elif SECTION == 3

    // Initial conditions
    double Y[14] = {0.0, 0.0, 0.0, 0.0, 0.1, 0.0, -1.0, 1.0, 0.0, 0.1, 0.0, 0.0, 0.0,
    1.0};

    // Open the output file
    fdata.open("particle_3.dat");

    // Loop to perform integration steps
    for (double t = t0; t <= tf; t += dt) {
        // Print the current time and state to the output file
```

```
        fdata << t << " ";
        for (int i = 0; i < Neq; i++) {
            fdata << Y[i] << " ";
        }
        fdata << sqrt(Y[4]*Y[4] + Y[3]*Y[3]) << endl;

        // Call the Runge-Kutta method to update the state
        RK4(t, Y, dYdt, dt, Neq);
    }
    fdata.close();

    // Analytical solution using a new system X' in motion with v_x = 0.1
    double x, y;
    double v_p;       // Velocity of the particle in the system X'
    double B_p = Y[13] + 0.1 * Y[9]; // Magnetic field in the system X'
    double r_L = v_p/1.01; //Larmor radius in the system x'
    t = 0.0;
    v_p = 0.1 * sqrt(2.0);

    fdata.open("analytical_trajectory_2.dat");
    for (i = 0; i <= 200; i ++) {
        // Compute the current time and state to the output file
        x = r_L * cos(t + M_PI/4.0) - r_L/sqrt(2.0) + 0.1 * t;
        y = r_L * sin(t + M_PI/4.0) - r_L/sqrt(2.0);
        fdata << t << " " << x << " " << y << endl;
        t += dt;
    }
    fdata.close();

    // Compute the error for t from 0 to 20 s
    fdata.open("error_RK.dat");
    // Reset initial conditions
    Y[0] = Y[1] = Y[2] = Y[3] = 0.0;
    Y[4] = Y[5] = 0.1;
    t = t0;

    for (i = 0; i <= 200; i++){
        // Print the error at current time
        x = r_L * cos(t + M_PI / 4.0) - r_L / sqrt(2.0) + 0.1 * t;
        fdata << t << " " << fabs(Y[0] - x) << endl;

        // Call the RK4 function to update the state and update time
        RK4(t, Y, dYdt, dt, Neq);
        t += dt;
    }
    fdata.close();

    #elif SECTION == 4

    int N = 3;
    int p;
    double x[N], v[Neq]; // Position and Velocity
    double E[N] = {0.0, 0.0, 0.5}; //Electic field
    double B[N];   // Magnetic fields
    double v0 = 0.1;
    double energy; //total energy of the particle
    srand48(time(NULL));

    fdata.open("particle_1000.dat");
    flog.open("energy.dat");

    // Loop over p=1000 particle
    for(p=0; p<1000; p++){
        // Define random angles theta in [0, Pi] and phi in [0, 2Pi]
        double theta, phi;
        theta = drand48() * M_PI;
        phi = drand48() * 2 * M_PI;

        // Define position and velocity of one particle
```

```
201         x[0] = (drand48() - 0.5) * 2000.0;
202         x[1] = (drand48() - 0.5) * 2000.0;
203         x[2] = 0.0;
204 
205         v[0] = v0 * sin(theta) * cos(phi);
206         v[1] = v0 * sin(theta) * sin(phi);
207         v[2] = v0 * cos(theta);
208 
209         // Define initial Magnetic fields
210         B[0] = 0.5 * x[1];
211         B[1] = 0.5 * x[0];
212         B[2] = 0.0;
213 
214         for (double t = t0; t<tf; t+=dt){
215 
216            // Print current state in the output file
217            fdata << p << " " << t << " " << x[0] << " " << x[1] << " " << x[2];
218            fdata << " " << v[0] << " " << v[1] << " " << v[2] << endl;
219 
220            // Call the Boris integrator
221            Boris(x, v, E, B, dt, N);
222 
223            // Compute the magnetic field for the new position
224            B[0] = x[1] / 1000.0;
225            B[1] = x[0] / 1000.0;
226            B[2] = 0.0;
227         }
228         fdata << endl;
229         fdata << endl;
230 
231         // Compute the energy of the particle and write it to an output file
232         energy = 0.5 * v[0] * v[0] + v[1] * v[1] + v[2] * v[2];
233         flog << energy << endl;
234     }
235 
236     fdata.close();
237     flog.close();
238     #endif
239 
240     return 0;
241 
242 }
243 
244 void RK4(double t, double * Y, void(*RHSFunc)(double, double *, double *), double dt,
        double Neq){
245     int i;
246     double k1[14], k2[14], k3[14], k4[14], Y1[14], Y2[14], Y3[14];
247 
248     RHSFunc(t, Y, k1); // compute k1
249     for(i = 0; i < Neq; i++) Y1[i] = Y[i] + 0.5 * dt * k1[i];
250 
251     RHSFunc(t + 0.5 * dt, Y1, k2); // compute k2
252     for(i = 0; i < Neq; i++) Y2[i] = Y[i] + 0.5 * dt * k2[i];
253 
254     RHSFunc(t + 0.5 * dt, Y2, k3); // compute k3
255     for(i = 0; i < Neq; i++) Y3[i] = Y[i] + dt * k3[i];
256 
257     RHSFunc(t + dt, Y3, k4); // compute k4
258     for(i = 0; i < Neq; i++) Y[i] += dt * (k1[i] + 2
259      * k2[i] + 2 * k3[i] + k4[i]) / 6.0; //final result of the differential equation
260 }
261 
262 void dYdt (double t, double *Y, double *R){
263     double x  = Y[0];
264     double y  = Y[1];
265     double z  = Y[2]; //(x,y,z) = Position
266     double vx = Y[3];
267     double vy = Y[4];
268     double vz = Y[5]; // (vx, vy, vz) = Velocity
```

```
    double q  = Y[6]; // Charge
    double m  = Y[7]; // Mass
    double Ex = Y[8];
    double Ey = Y[9];
    double Ez = Y[10]; // Electric Field
    double Bx = Y[11];
    double By = Y[12];
    double Bz = Y[13]; // Magnetic Field


    R[0] = vx; // Solution for each Variable dY/dt= R(Y,t)
    R[1] = vy;
    R[2] = vz;
    R[3] = (q / m) * (Ex + vy * Bz - vz * By);
    R[4] = (q / m) * (Ey + vz * Bx - vx * Bz);
    R[5] = (q / m) * (Ez + vx * By - vy * Bx);
    R[6] = 0.0;
    R[7] = 0.0;
    R[8] = 0.0;
    R[9] = 0.0;
    R[10]= 0.0;
    R[11]= 0.0;
    R[12]= 0.0;
    R[13]= 0.0; // Each constant is put in the form of (dy/dt=0), y=constant

}

void Boris(double *x, double *v, double *E, double *B, double dt, int N){
    int i;
    double m = 1.0;   // Mass
    double q = -1.0;  // Charge
    double gamma;     // Lorentz factor
    double h = (q / m) * dt;
    double b_squared;
    double v_squared;

    double b[N];
    double u_minus[N];
    double u_plus[N];
    double u[N];
    double uxb[N];

    //Current velocity squared and Lorentz factor
    v_squared = v[0] * v[0] + v[1] * v[1] + v[2] * v[2];
    gamma = 1.0; // we define gamma = 1 for the non relativistic formula.
                 // For the relativistic case gamma = sqrt(1.0/(1.0-v_squared));

    //Modify the magnetic field vector
    for (i = 0; i < N; i++) b[i] = h * 0.5 * B[i] / gamma;

    // Square the modified magnetic field vector
    b_squared = b[0] * b[0] + b[1] * b[1] + b[2] * b[2];

    // Compute position using current velocity
    for (i=0; i< N; i++) x[i] += 0.5 * dt * v[i];

    // Compute velocity using acceleration generated by electric field (kick)
    for (i=0; i< N; i++) u_minus[i] = v[i] * gamma + 0.5 * h * E[i];

    // Compute the cross product (u_minus x b)
    uxb[0] = u_minus[1] * b[2] - u_minus[2] * b[1];
    uxb[1] = u_minus[2] * b[0] - u_minus[0] * b[2];
    uxb[2] = u_minus[0] * b[1] - u_minus[1] * b[0];

    // Add the rotation due to magnetic field (rotation)
    u_plus[0] = u_minus[0] + 2*(uxb[0] + uxb[1]*b[2]-uxb[2]*b[1])/(1+b_squared);
    u_plus[1] = u_minus[1] + 2*(uxb[1] + uxb[2]*b[0]-uxb[0]*b[2])/(1+b_squared);
    u_plus[2] = u_minus[2] + 2*(uxb[2] + uxb[0]*b[1]-uxb[1]*b[0])/(1+b_squared);

```

```
    // Compute current velocity and position using electric field and velocity
    for (i=0; i< N; i++) v[i] = (u_plus[i] + 0.5 * h * E[i]) / gamma;
    for (i=0; i< N; i++) x[i] += 0.5 * dt * v[i];
}
```

Per il codice usato nell'ultima sessione (Boris relativistico) abbiamo definito correttamente il fattore di Lorentz invece di porre $\gamma = 1$. L'unica linea di codice modificata è la seguente:

```

    gamma = sqrt(1.0/(1.0-v_squared));
```

Di seguito sono riportati i codici Gnuplot per creare gli istogrammi e fittare le curve.

Codice per creare l'istogramma partendo da un file con una singola colonna contenente i valori delle energie:

```
clear
# Define the energy range and bin width
bin_width = 0.2
min_energy = 0.0
max_energy = 10

# Calculate the number of bins
num_bins = int((max_energy - min_energy) / bin_width)

# Set the x-axis label and title
set xlabel "Energy"
set ylabel "Frequency"
set title "Particle Energy Histogram"

# Set the style and range for the x-axis
set style fill solid
set xrange [min_energy+0.099:max_energy]
set yrange [0.1:300]
set xtics min_energy, bin_width*5, max_energy


# Create the histogram
bin(x) = floor(x / bin_width) * bin_width + bin_width/2.0
plot 'energy.dat' using (bin($1)):(1.0) smooth frequency with points lc rgb "blue"
    notitle
```

Codice per produrre un file .dat che contenga una colonna con il numero di particelle in un certo range di energia e una colonna con i valori centrali degli intervalli di energia:

```
# Define the bin width and range of energies
bin_width = 0.2
min_energy = 0.0
max_energy = 10

# Calculate the number of bins and set the range for the histogram
num_bins = int((max_energy - min_energy) / bin_width)
set xrange [min_energy : max_energy]

# Define the shell command to calculate the histogram with awk
bin_command = sprintf("awk -v bin_width=%f '{ bin=int($1 / bin_width); hist[bin] += 1; }
    END { for (i in hist) { print i * bin_width + bin_width/2.0, hist[i]; } }' energy.dat
    ", bin_width)

# Load data using the shell command and store it in an internal dataset
plot "<".bin_command using 1:2 with linespoints title "Histogram"

# Save the histogram data to a new two-column .dat file
output_file = "binned_data.dat"
set table output_file
plot "<".bin_command using 1:2 with table
unset table
```

Codice per fittare i dati contenuti nel file .dat a due colonne con la power law:

```
clear
# Define the power law function
f(x) = a * x**b

# Set initial guess for parameters a and b
a = 100.0
b = -1.5

# Perform the fit
fit f(x) 'binned_data.dat' using 1:2 via a, b

# Print the fitting results
print "Fitting Results:"
print "a =", a
print "b =", b

# Plot the data points and the fitted curve
set xlabel "x"
set ylabel "y"
set title "Power Law Fit"
plot 'binned_data.dat' using 1:2 with points title "Data", f(x) with lines title "Fit"
```