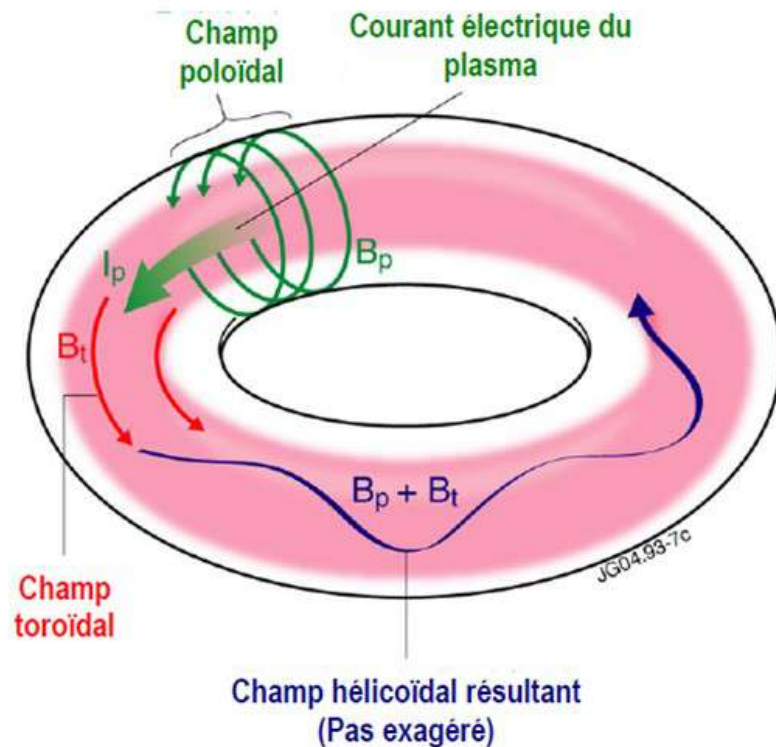


Confinement magnétique de magma : Un nouvel atout pour les énergies du futur

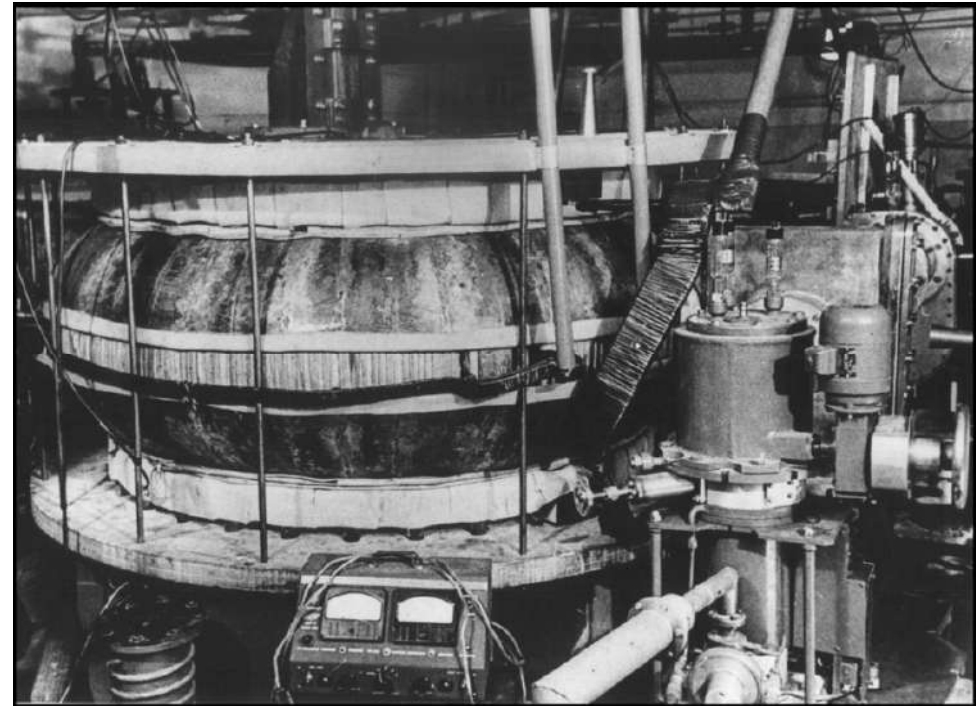


Tokamak

Champ magnétique à l'intérieur du tokamak

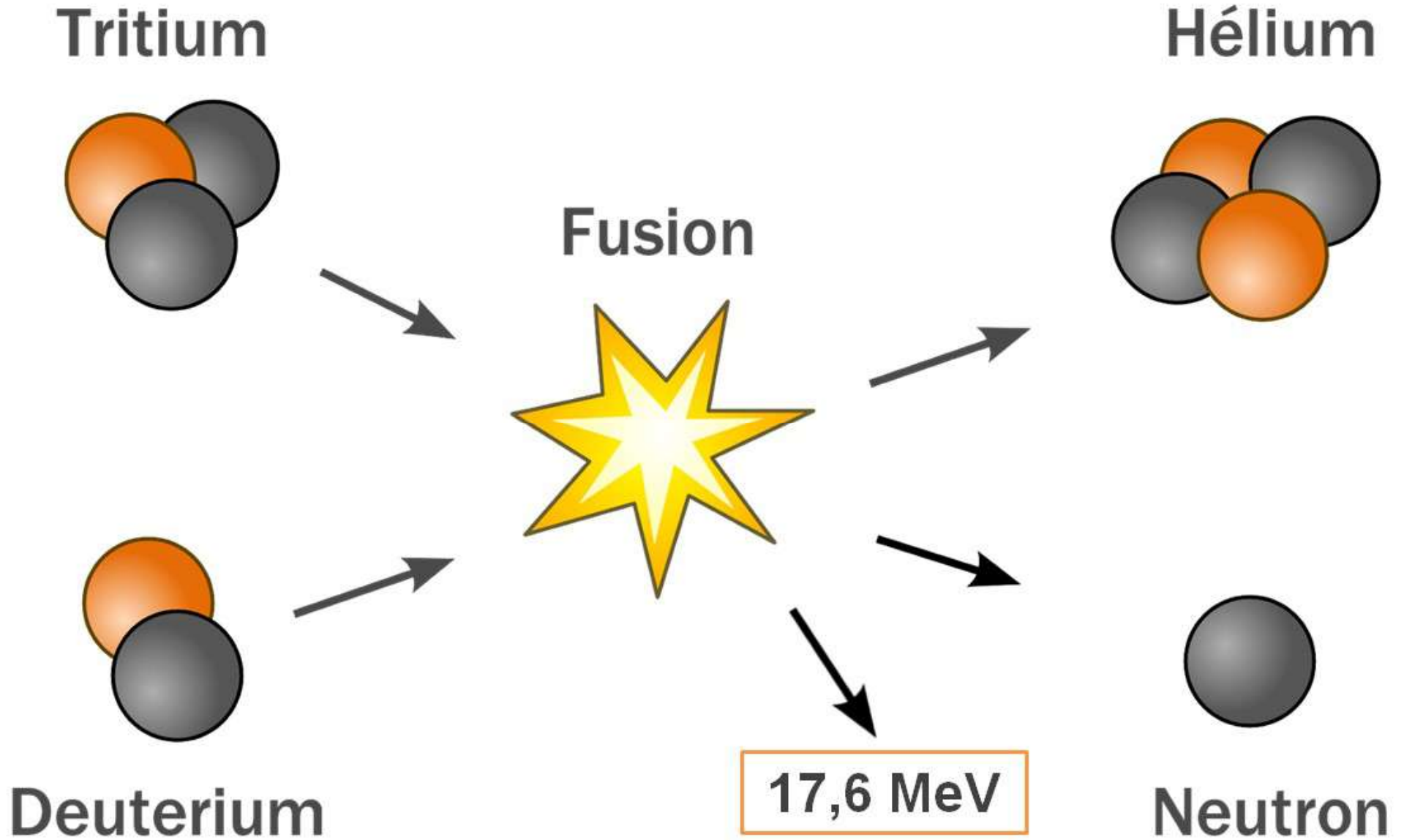


Tokamak T1 à l'institut Kourtchatov à Moscou

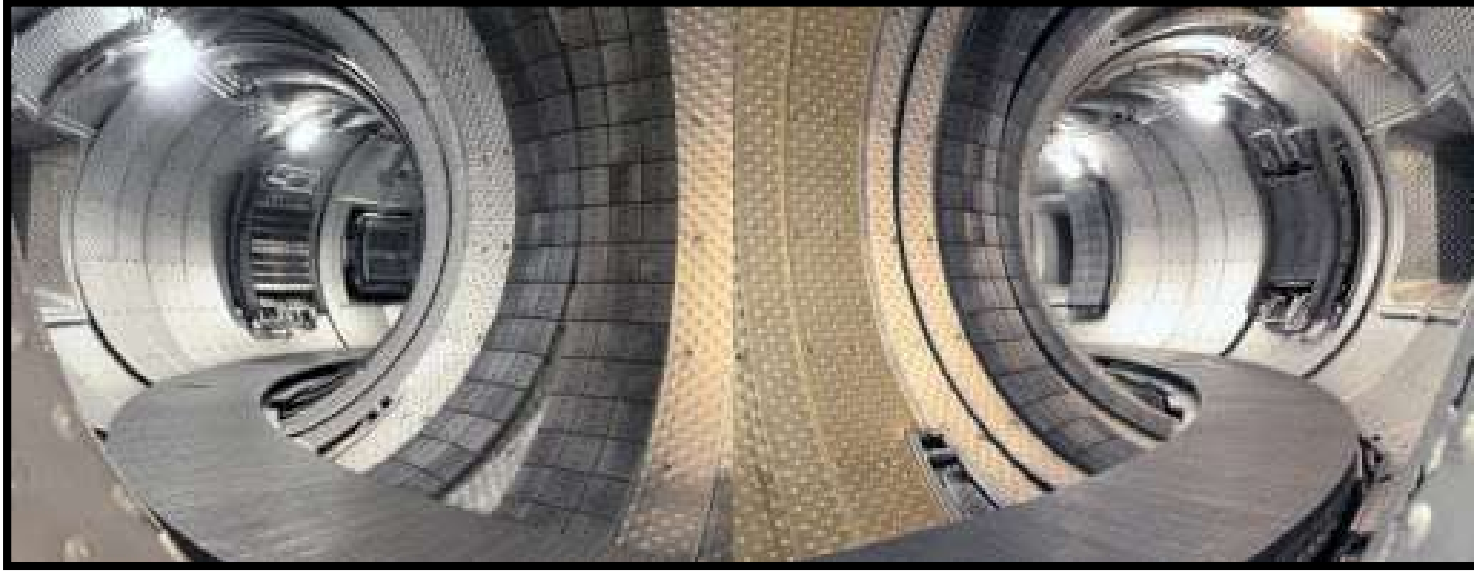


Issu du russe: *toroidalnaïa kamera s magnitnymi katouchkami*, qui signifie: **chambre toroïdale avec bobines magnétiques**

Réaction de fusion nucléaire

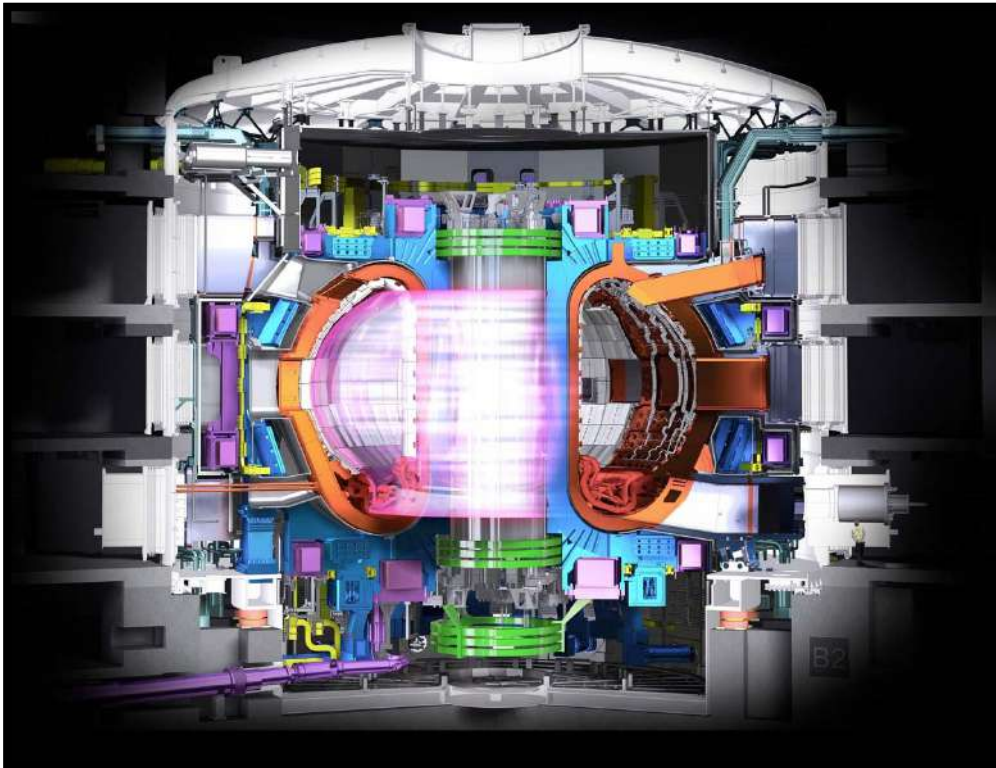


Tokamak Tore Supra



Tokamak ITER

(International Thermonuclear Experimental Reactor)

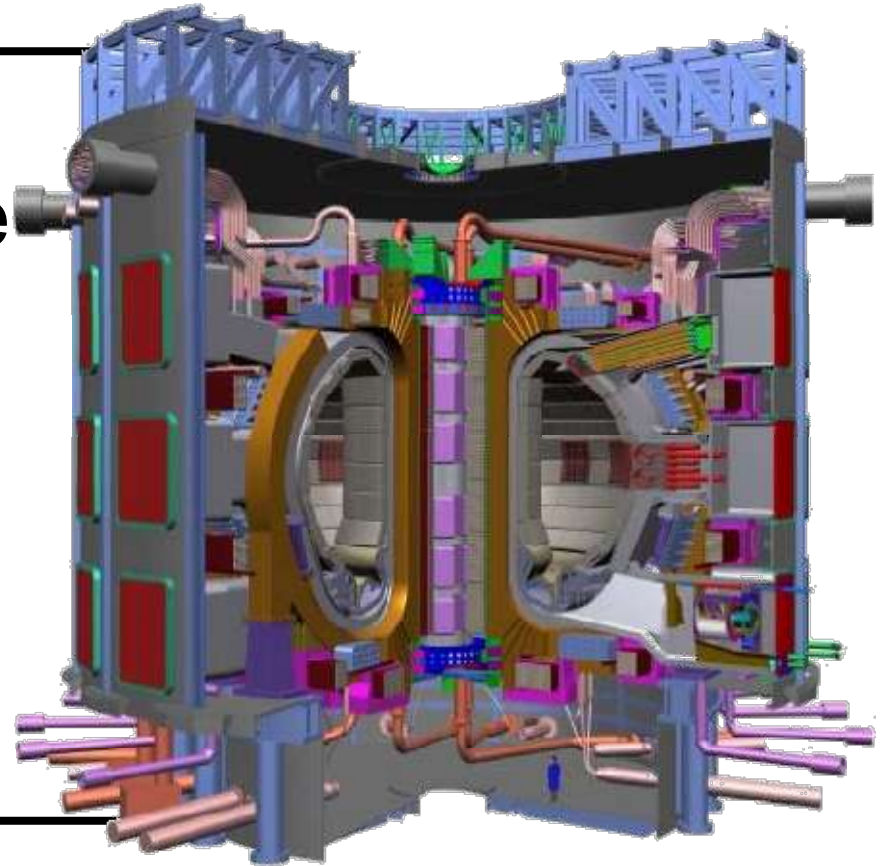


Plan:

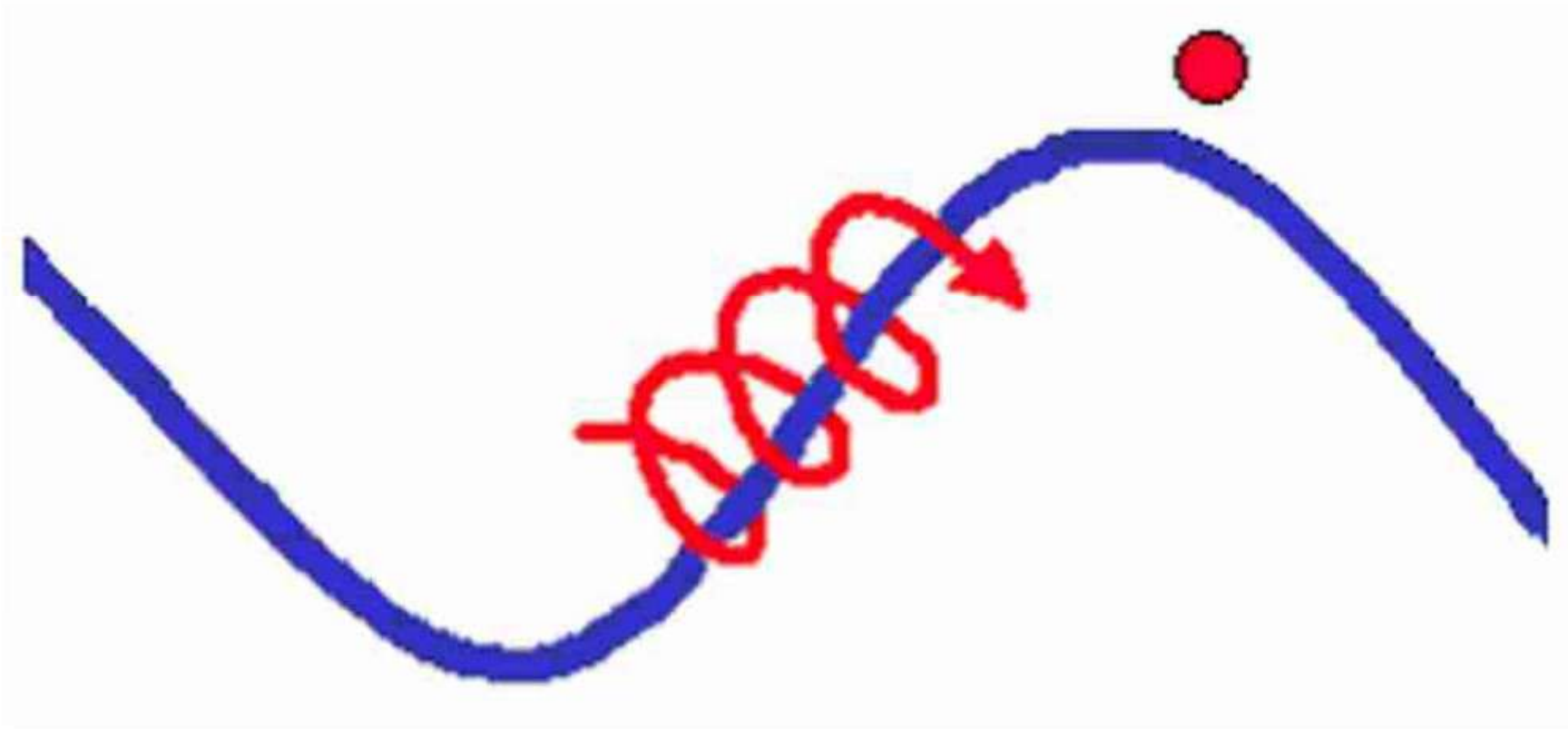
I: Confinement Magnétique

II: Etude d'un Tokamak

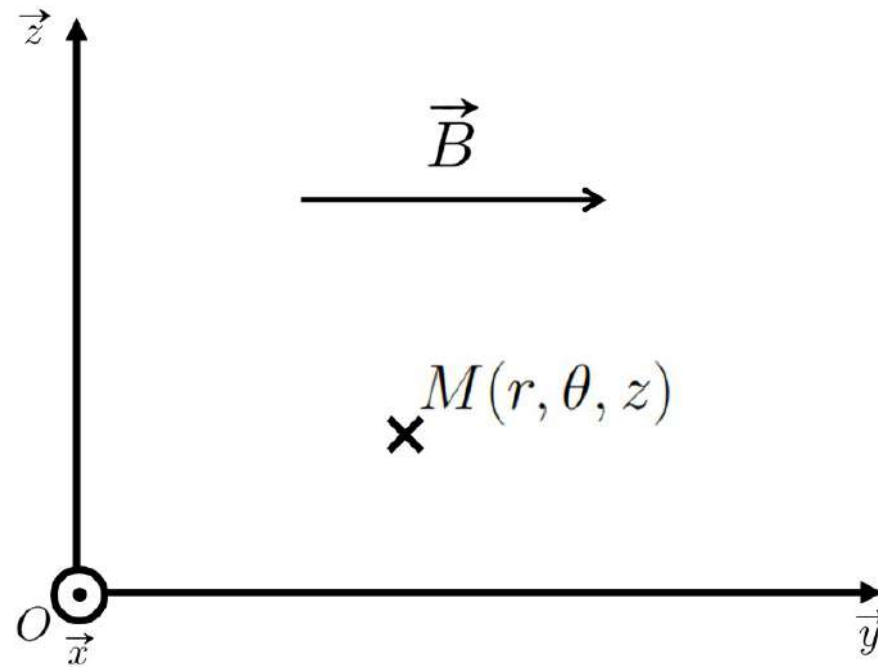
III: Amélioration du Modèle



I: Confinement Magnétique



Etude dans un champ uniforme



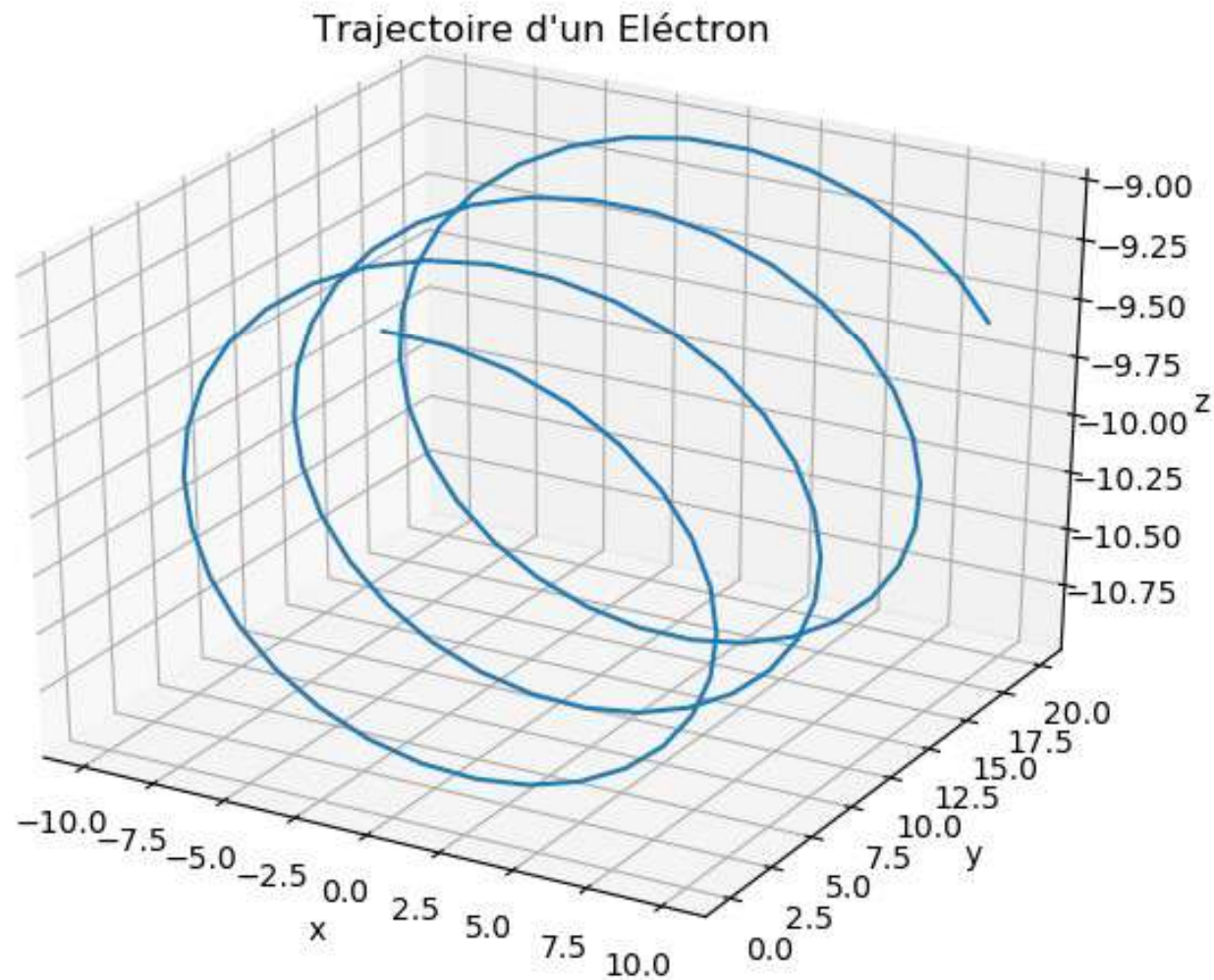
Système: Particule chargée de charge q et de masse m

Bilan des forces: Force de Lorentz: $\vec{F} = q\vec{v} \wedge \vec{B}$

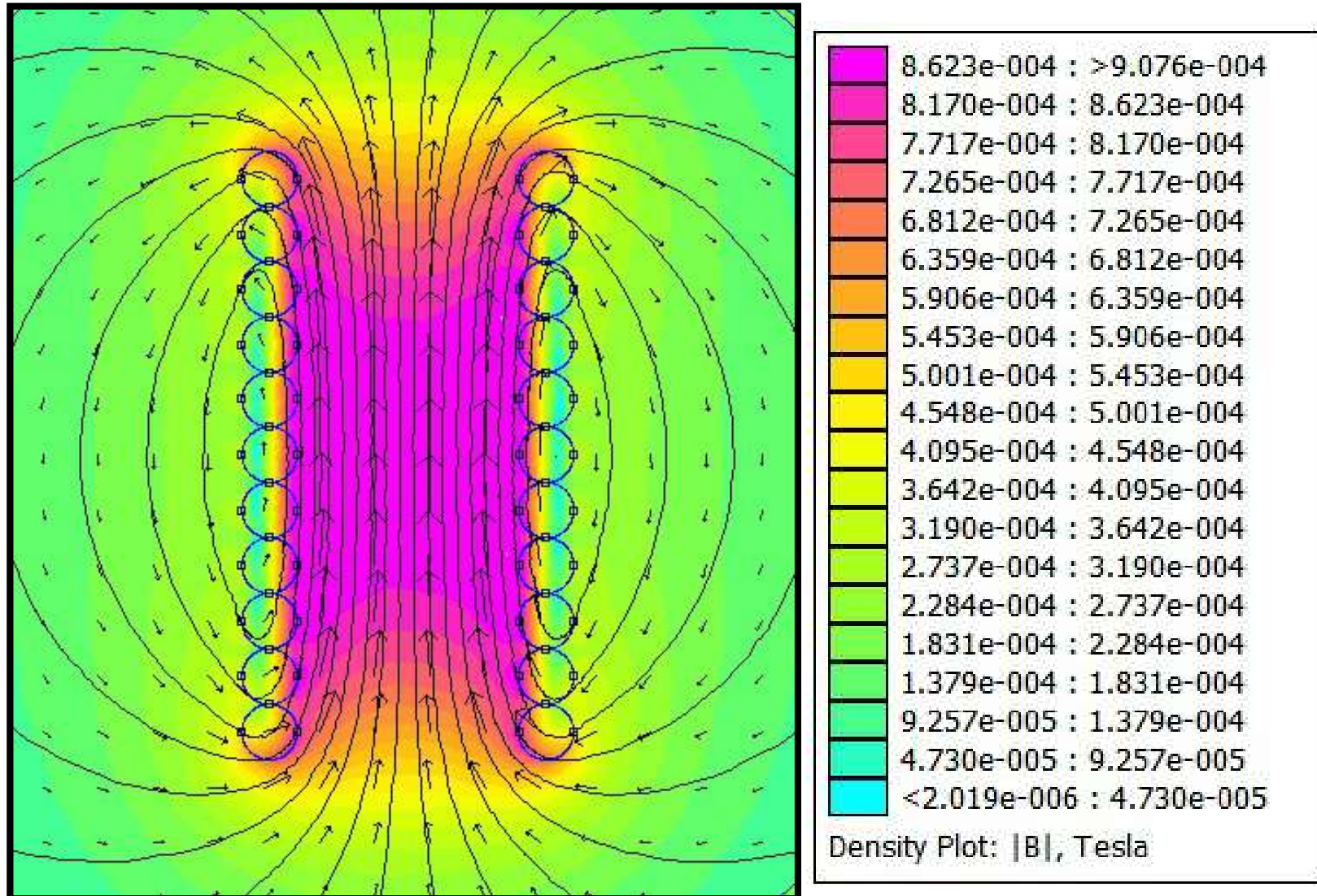
Principe fondamental de la dynamique: $m\frac{d\vec{v}}{dt} = q\vec{v} \wedge \vec{B}$

$$\begin{cases} x(t) = \rho_L \sin(\omega_c t) + x_0 \\ y(t) = v_{\parallel} t + y_0 \\ z(t) = \epsilon \rho_L (1 - \cos(\omega_c t)) + z_0 \end{cases}$$

Avec $\rho_L = \frac{v_{\perp}}{\omega_c}$ le rayon de Larmor

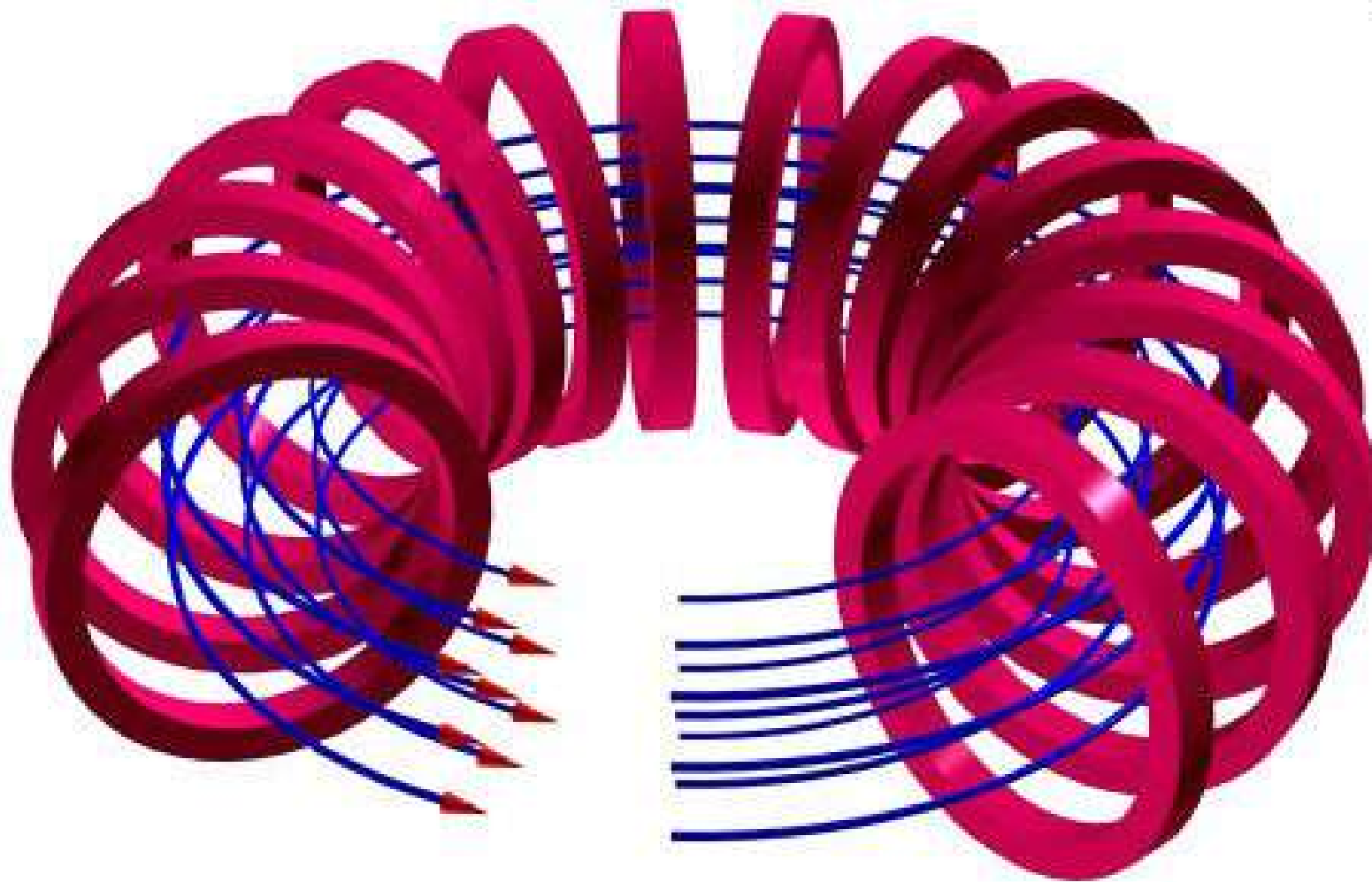


Représentation des lignes de champ d'un solénoïde avec le logiciel [FEMM](#)

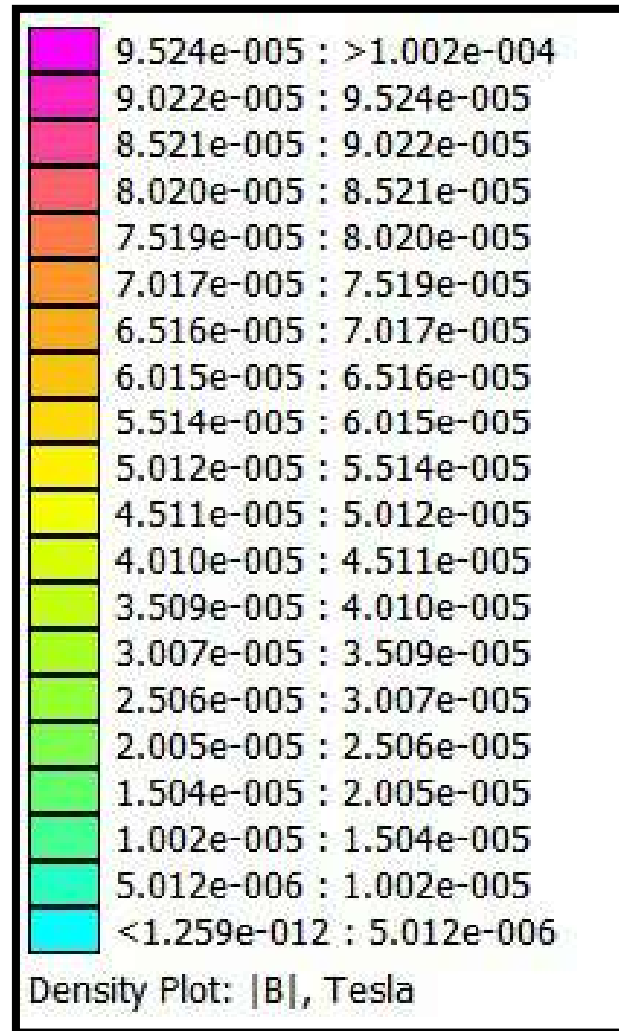
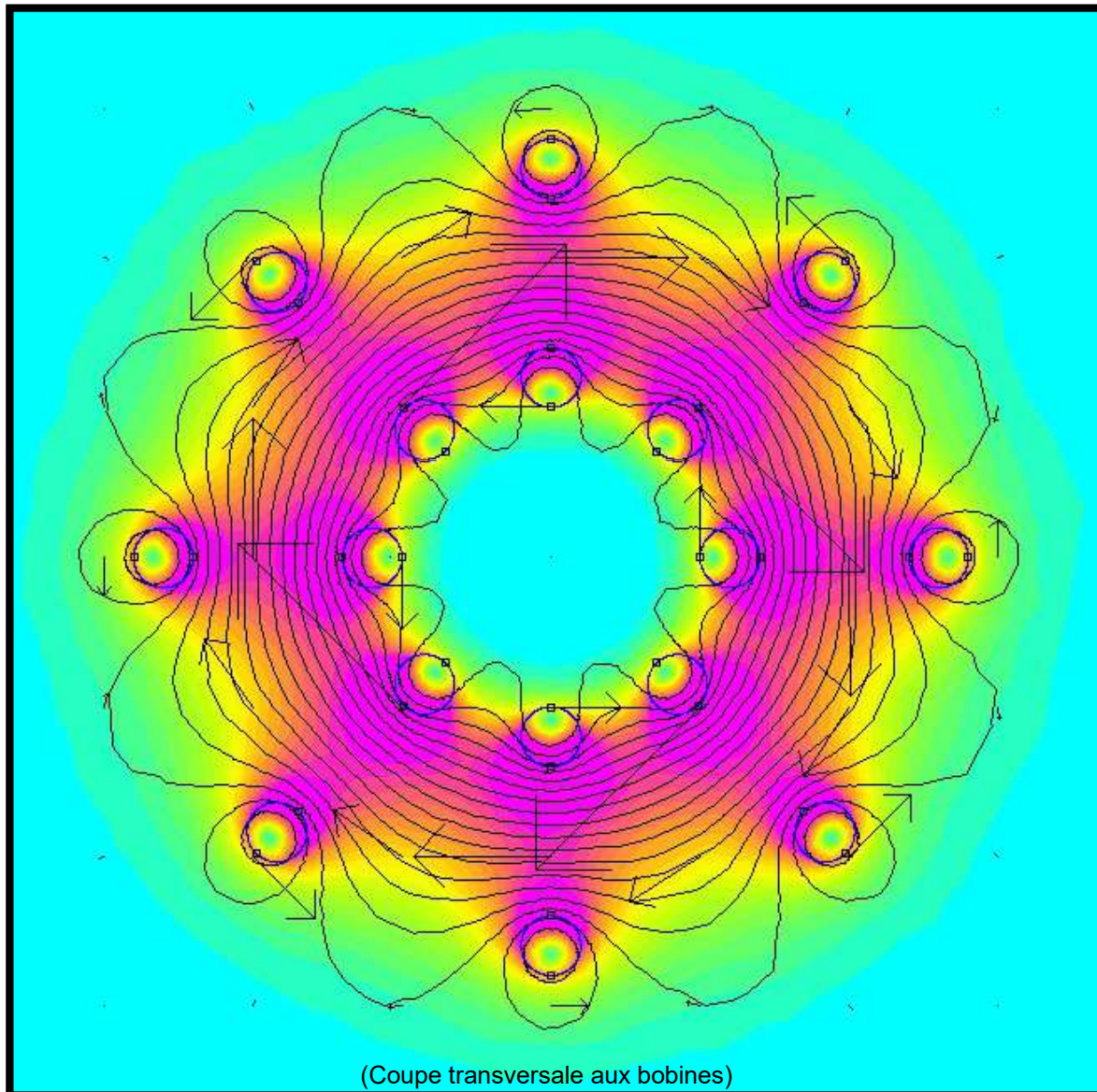


(Coupe transversale aux spires)

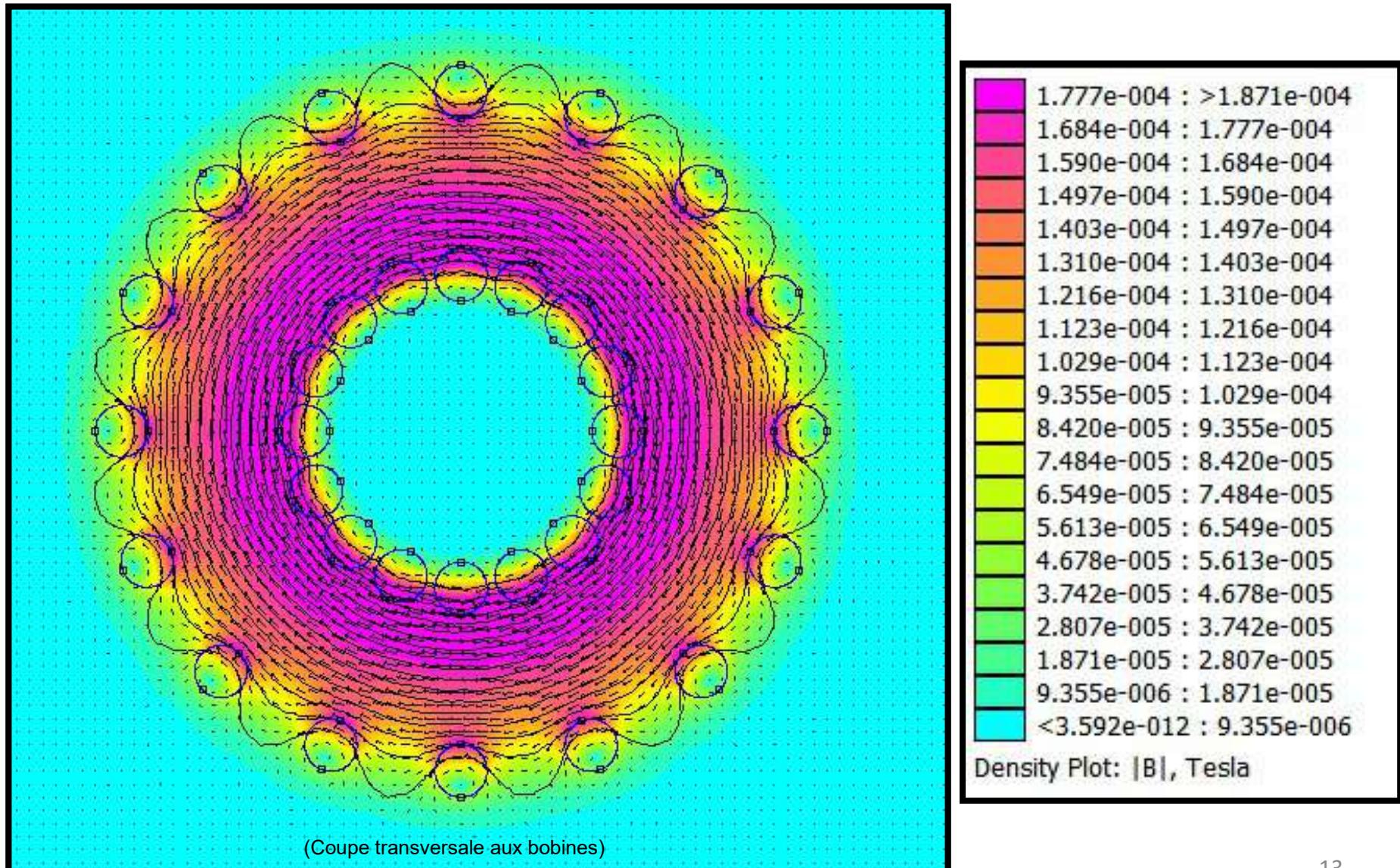
Schéma des lignes de champ dans un solénoïde torique



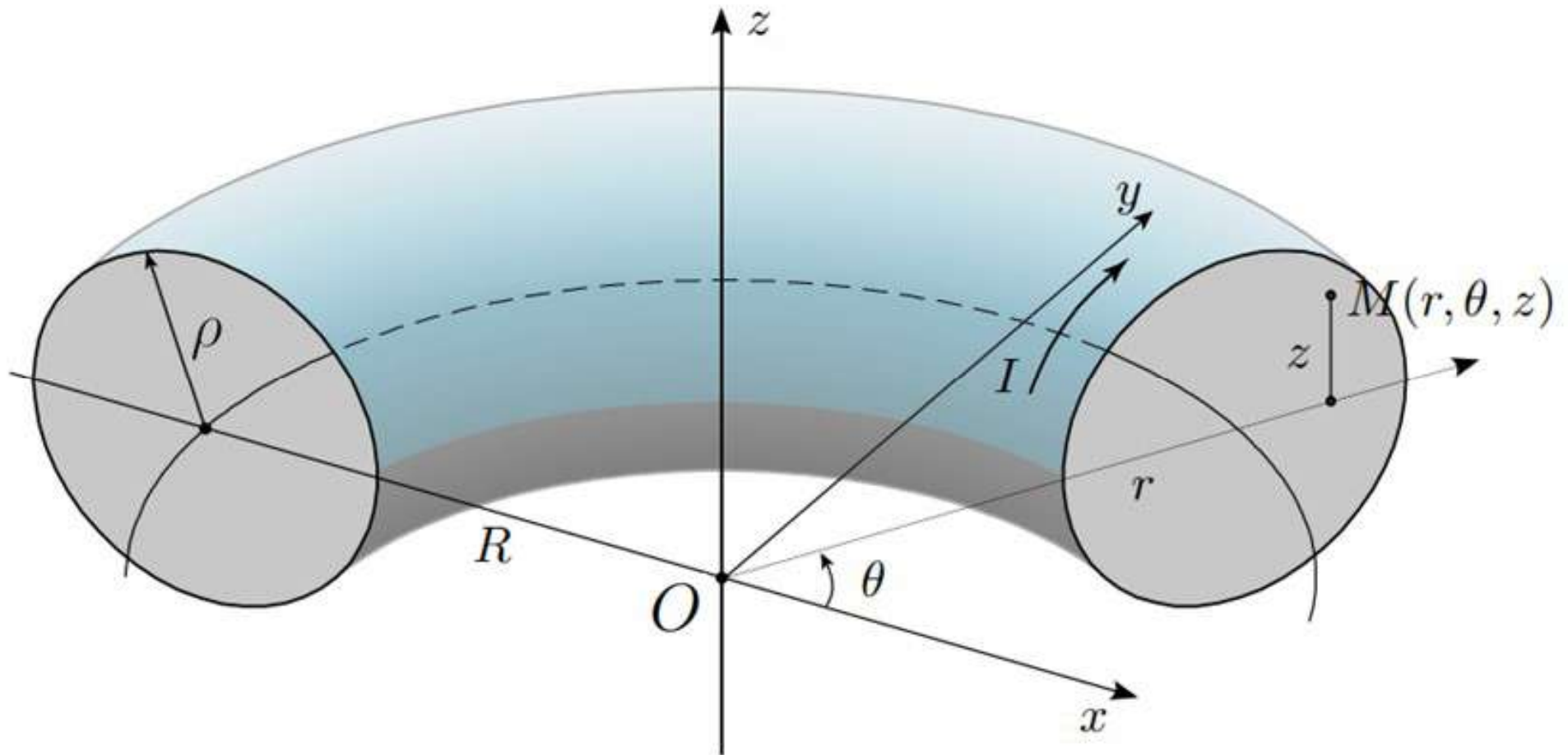
Représentation des lignes de champ d'un solénoïde torique avec un faible nombre de spires



Représentation des lignes de champ d'un solénoïde torique avec un nombre de spire plus important

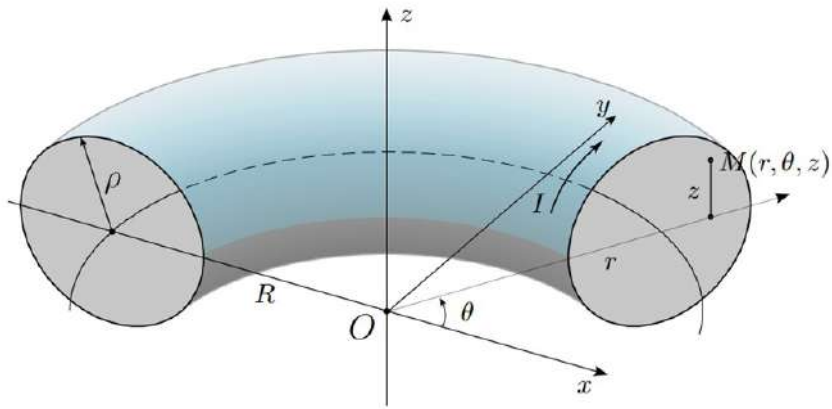


II: Etude d'un Tokamak

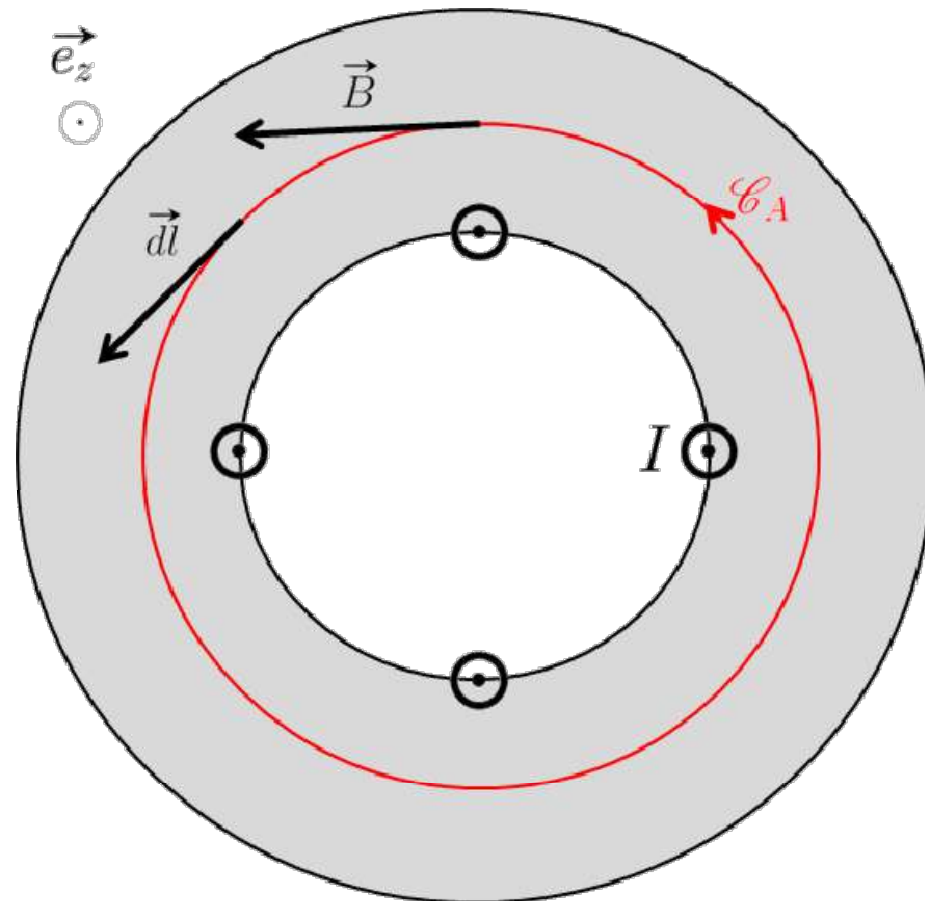


Représentation du tokamak assimilé à un tore et du système de coordonnées utilisé

D'après le théorème d'Ampère:



$$\vec{B} = \frac{\mu_0 N I}{2\pi r} \vec{e}_\theta$$



Système: Particule chargée de charge q et de masse m

Bilan des forces: Force de Lorentz: $\vec{F} = q\vec{v} \wedge \vec{B}$

Principe fondamental de la dynamique: $m \frac{d\vec{v}}{dt} = q\vec{v} \wedge \vec{B}$

$$(*) \quad \left\{ \begin{array}{l} m \left(\ddot{r} - r\dot{\theta}^2 \right) = -\frac{q\mu_0 NI}{2\pi r} \dot{z} \\ 2\dot{r}\dot{\theta} + r\ddot{\theta} = 0 \\ m\ddot{z} = \frac{q\mu_0 NI}{2\pi r} \dot{r} \end{array} \right.$$

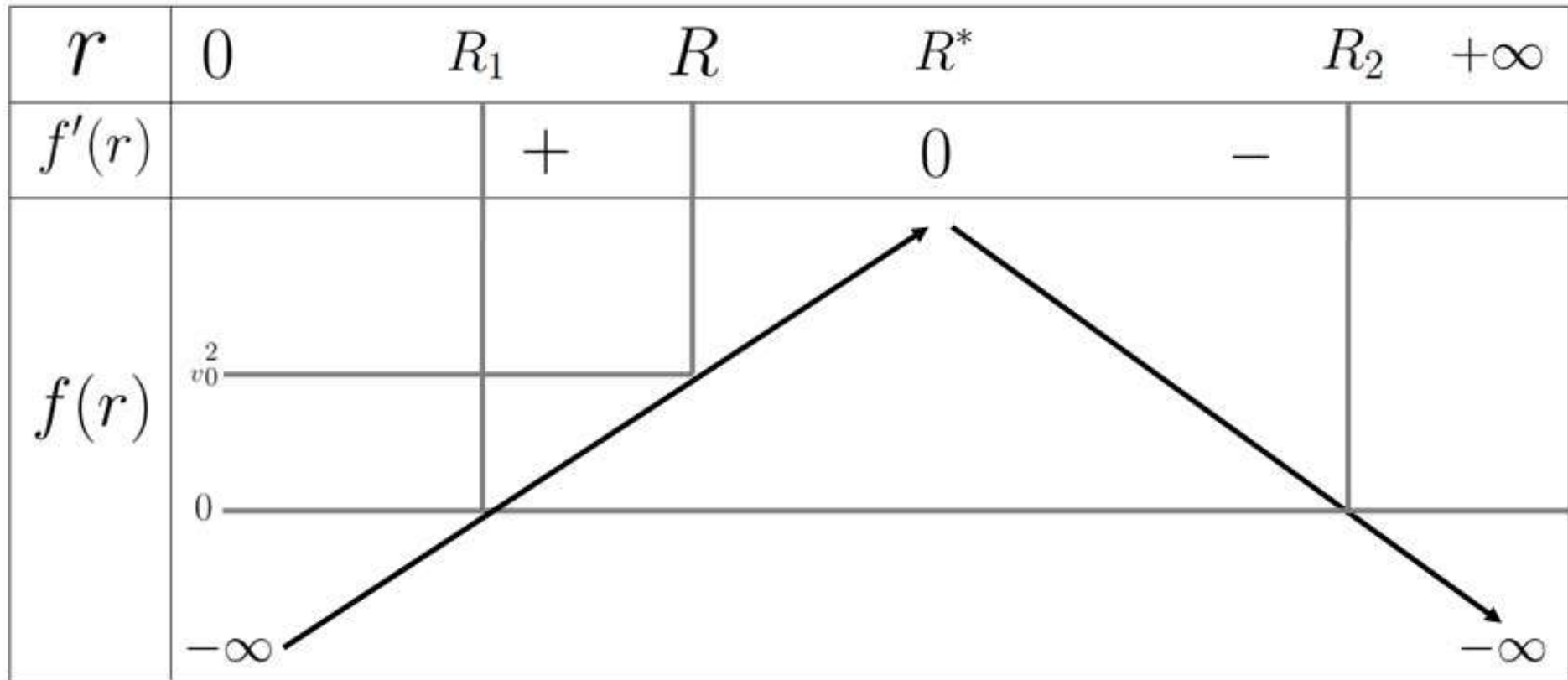
Conditions initiales:
$$\begin{cases} r_0 = R, \theta_0 = 0, z_0 = 0 \\ \dot{r}_0 = v_0, \dot{\theta}_0 = \frac{v_0}{R}, \dot{z}_0 = 0 \end{cases}$$

$$\dot{r}^2 = 2v_0^2 - \frac{(Rv_0)^2}{r^2} - \beta^2 \left(\ln \frac{r}{R} \right)^2 = f(r)$$

$$f'(r) = \frac{2}{r^3} (Rv_0)^2 - \beta^2 r^2 \ln \frac{r}{R}$$

Avec $\beta = \frac{q\mu_0 NI}{2\pi m}$

$$\dot{r}^2 = 2v_0^2 - \frac{(Rv_0)^2}{r^2} - \beta^2 \left(\ln \frac{r}{R} \right)^2 = f(r)$$



$$\dot{r}^2 = 2v_0^2 - \frac{(Rv_0)^2}{r^2} - \beta^2 \left(\ln \frac{r}{R} \right)^2 = f(r)$$

Possible présence de la particule

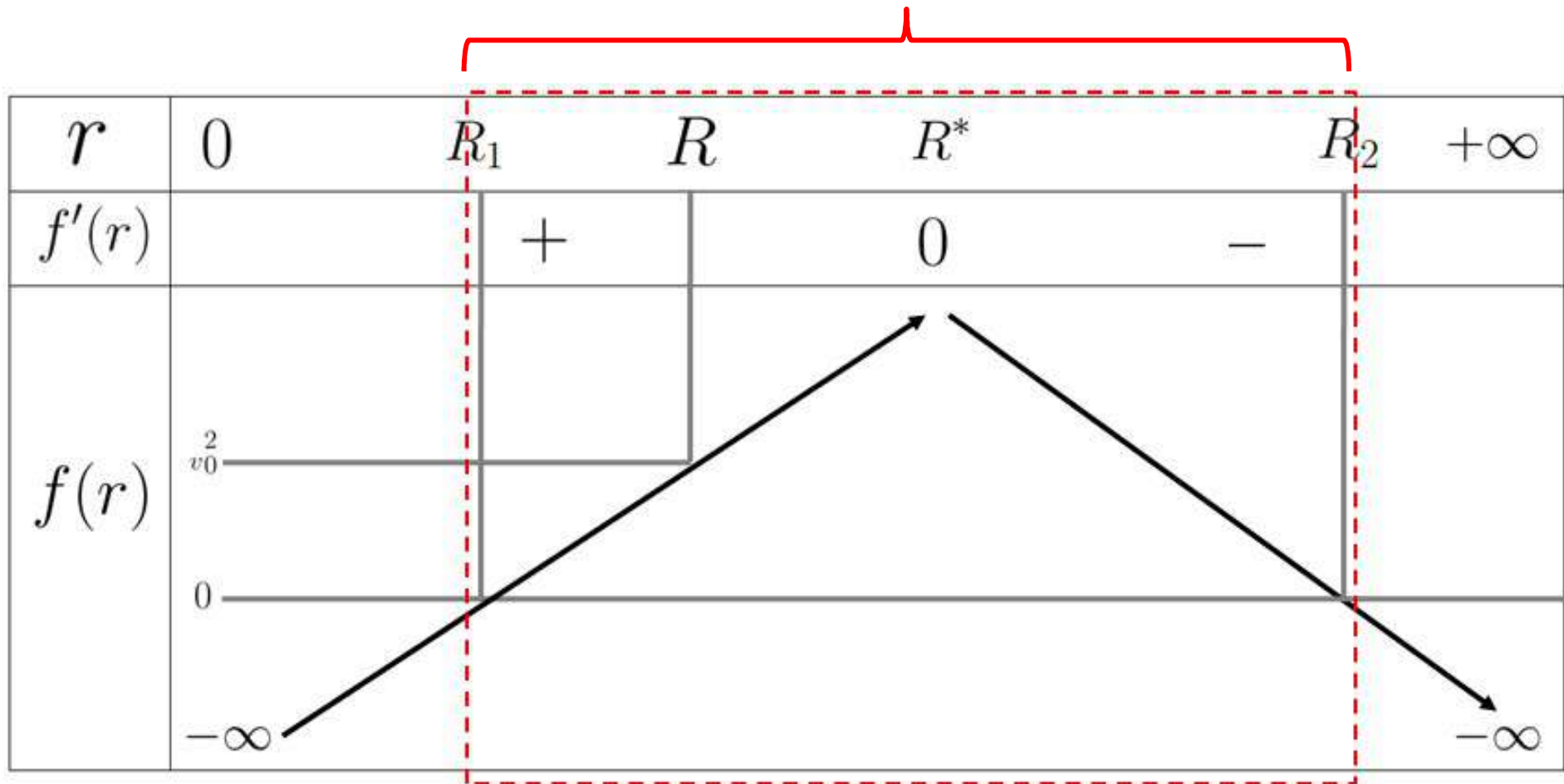


Tableau regroupant les caractéristiques de deux tokamaks

	ITER			TORE SUPRA			
	Doc.	Centrale	Tableau	Thèse	ESIM	Centrale	Tableau
Grand Rayon (R en m)	6,2	6,2	6,2	2,38	2,40	2,42	2,40
Petit Rayon (ρ en m)	2,0	2,0	2,0	0,8	1,20*	0,72 (du plasma)	0,7
Courant dans les Bobines (I en A)	80000 max dans les bobines toroïdales			1200	1400		
Nombre de Bobines (N)	18 Toroïdales/ 6 Poloidales/ 9 Correction/ 6 Centrales			18	18 (température 1,8 K)		
Nombre de spire par Bobine (n)					2028		
Champ Magnétique (B en T)	5,3	5,3	5,3	<4		4,5 (au centre)	4
Courant Plasma (I' en MA)	15	15	15	<2		1,7	1
Géométrie	Tore « Etiré selon Oz »			Tore « Circulaire »			

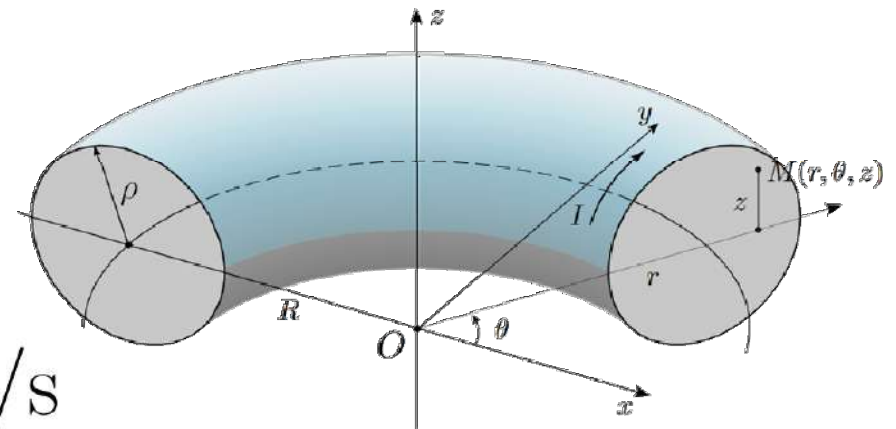
Choix des paramètres

Dimensions du tokamak: $R = 2,40$ m et $\rho = 0,70$ m

Nombre de Spires: $N = 18 \times 2028 = 36504$

Courant bobines: $I = 1400$ A

Vitesse initiale: $v_0 = 10^6$ m/s



On obtient: $R_1 = 2.399999$ et $R_2 = 2.400001$

Conditions initiales:
$$\begin{cases} r_0 = R, \theta_0 = 0, z_0 = 0 \\ \dot{r}_0 = v_0, \dot{\theta}_0 = \frac{v_0}{R}, \dot{z}_0 = 0 \end{cases}$$

(*) devient :

$$\begin{cases} \ddot{r} = \frac{(Rv_0)^2}{r^3} - \frac{\beta^2}{r} \ln \frac{r}{R} \\ \dot{\theta} = \frac{Rv_0}{r^2} \\ \dot{z} = \beta \ln \frac{r}{R} \end{cases}$$

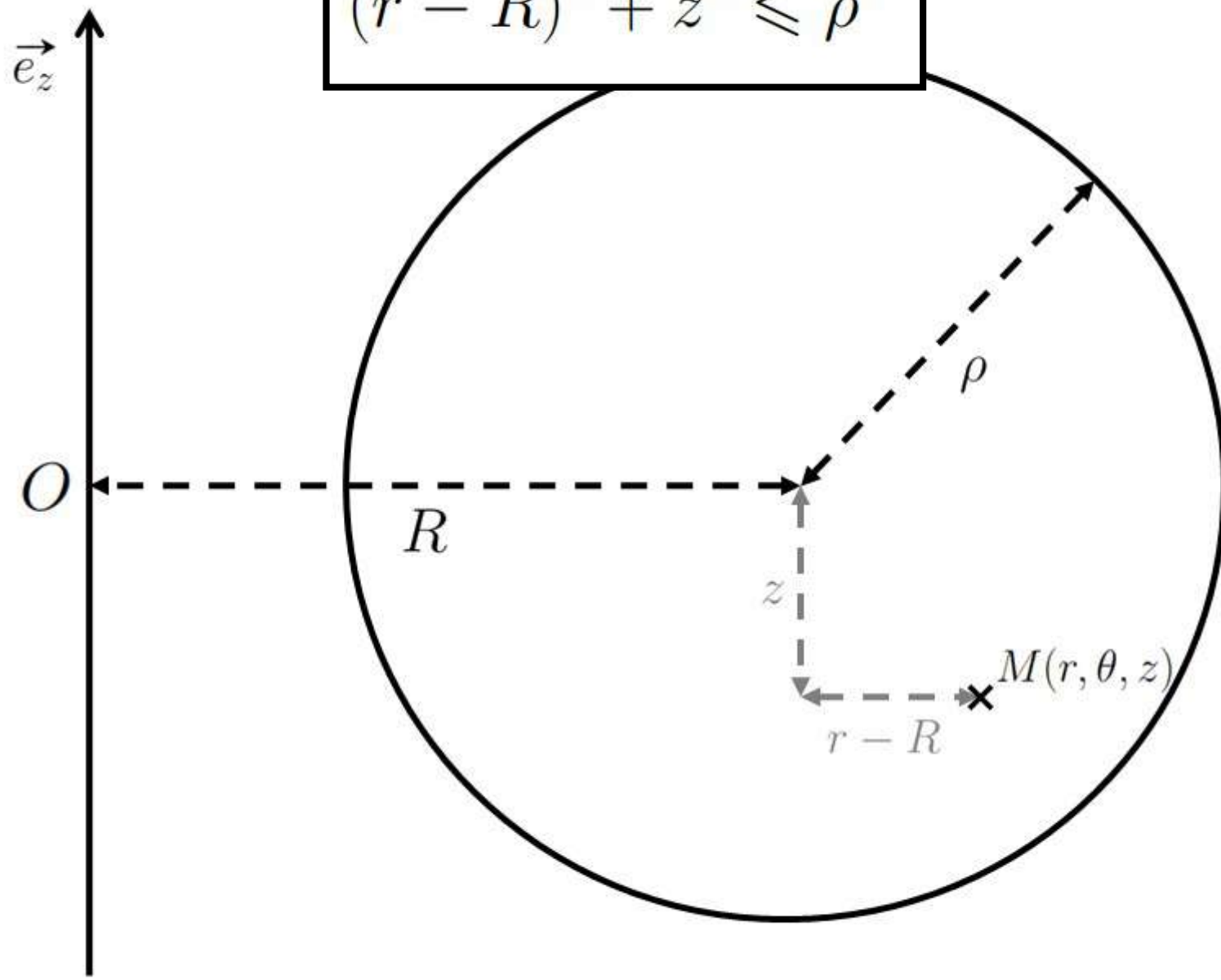
Mise en place d'une procédure numérique pour résoudre par approximation, des équations différentielles du premier ordre avec une condition initiale

$$\dot{Y} = \begin{pmatrix} \dot{r} \\ \dot{s} \\ \dot{\theta} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} s \\ \frac{(Rv_0)^2}{r^3} - \frac{\beta^2}{r} \ln \frac{r}{R} \\ \frac{Rv_0}{r^2} \\ \beta \ln \frac{r}{R} \end{pmatrix} = g(Y)$$

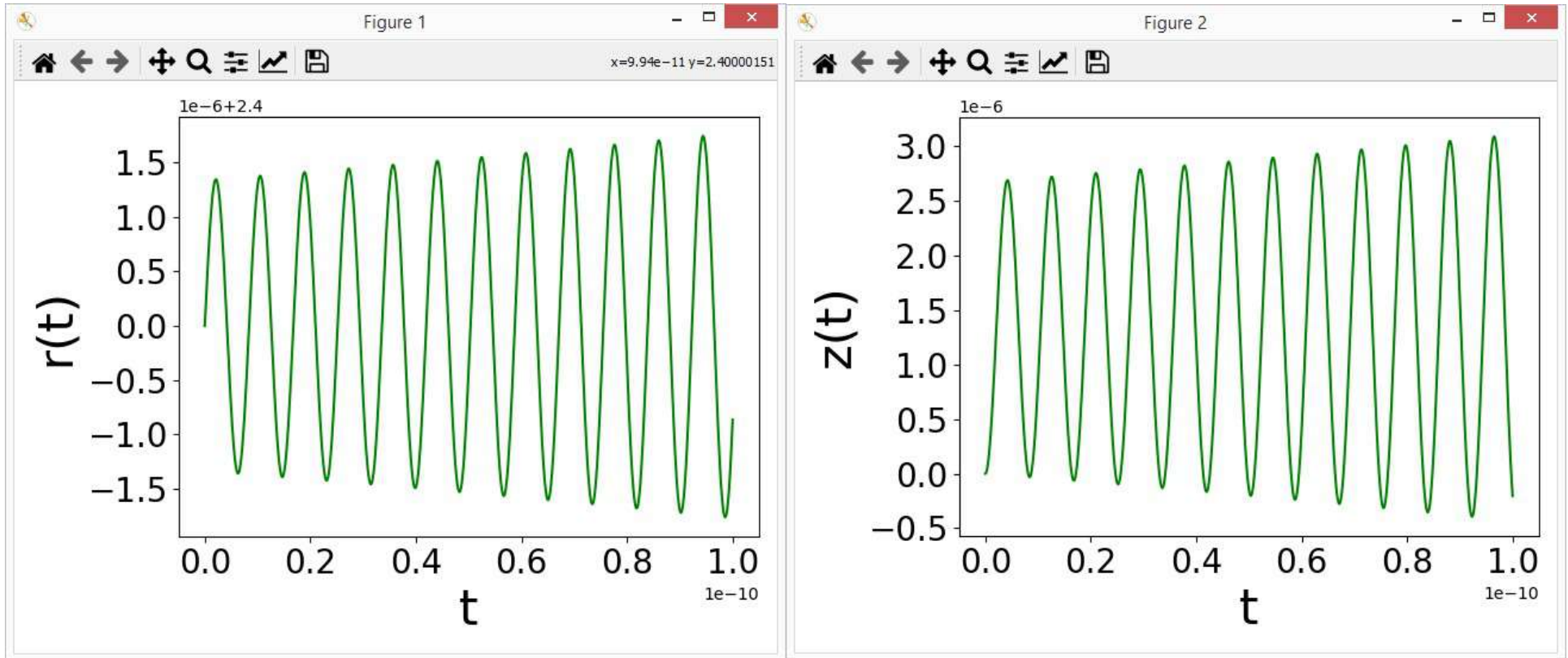
$$\begin{cases} r_{n+1} = r_n + \tau s_n \\ s_{n+1} = s_n + \tau \left(\frac{(Rv_0)^2}{r^3} - \frac{\beta^2}{r} \ln \frac{r}{R} \right) \\ \theta_{n+1} = \theta_n + \tau \frac{Rv_0}{r^2} \\ z_{n+1} = z_n + \tau \beta \ln \frac{r}{R} \end{cases}$$

Condition pour rester dans le tokamak

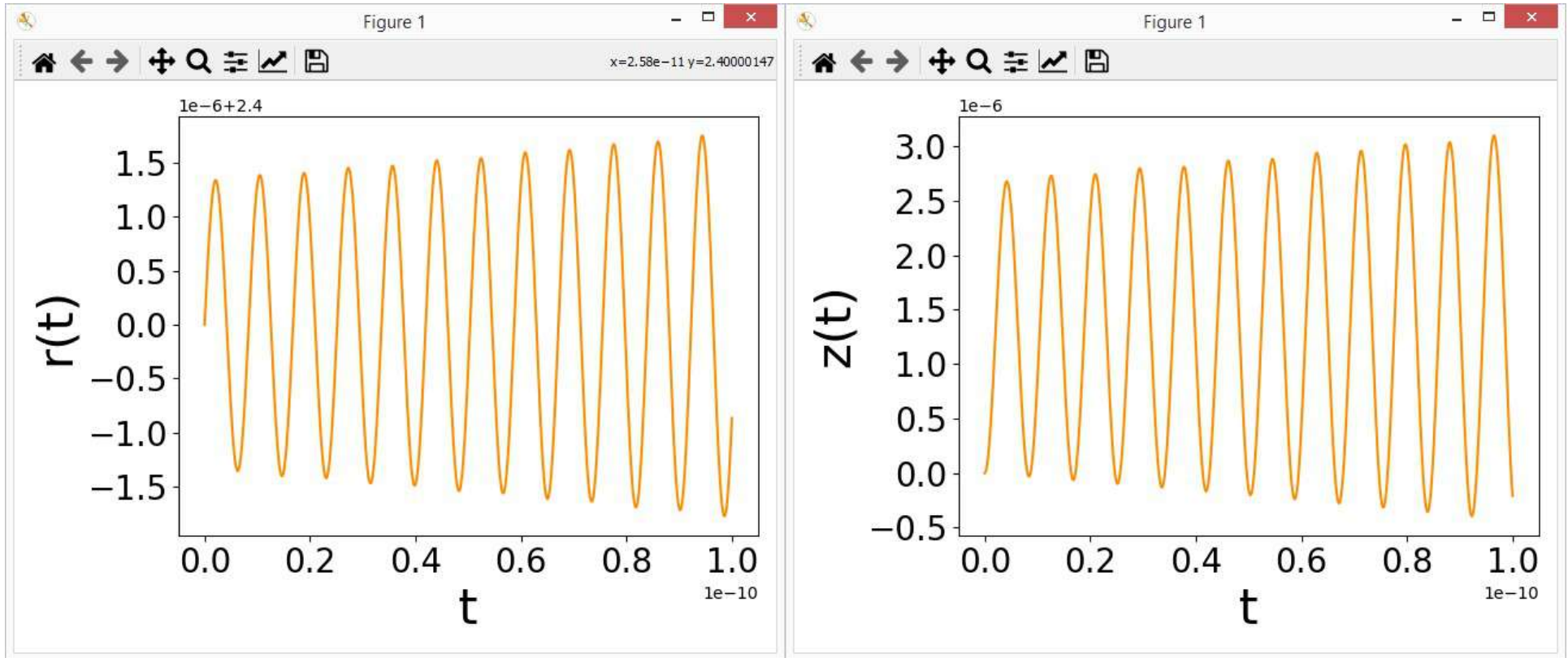
$$(r - R)^2 + z^2 \leq \rho^2$$



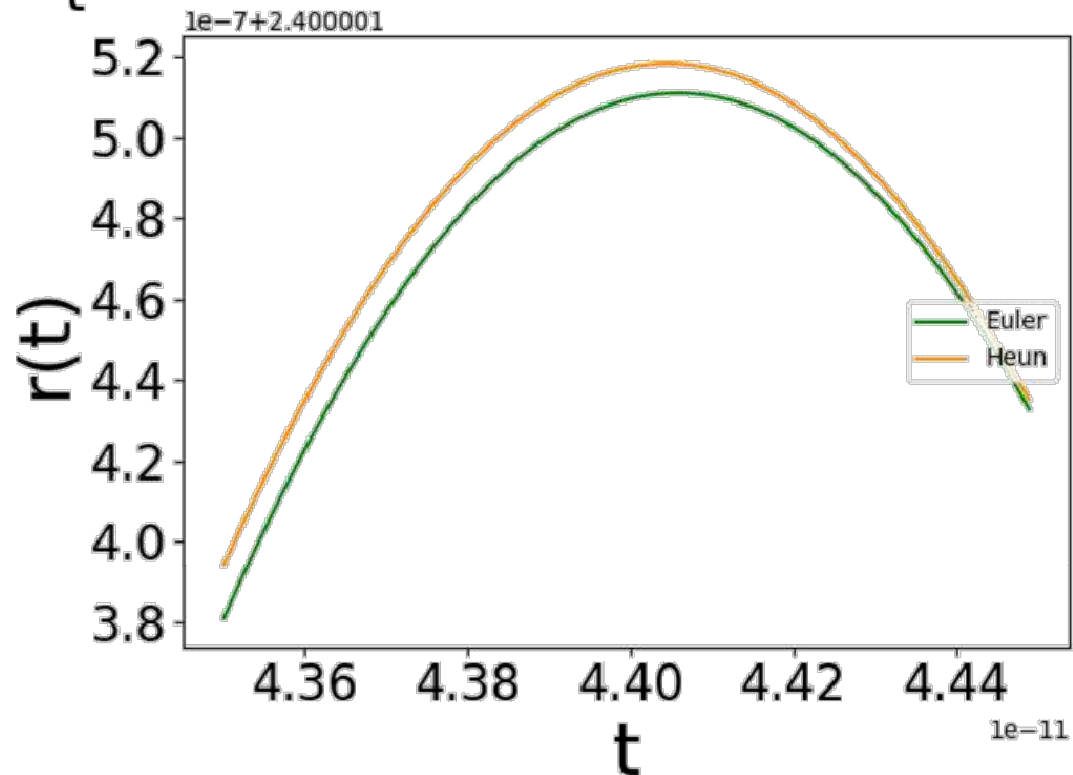
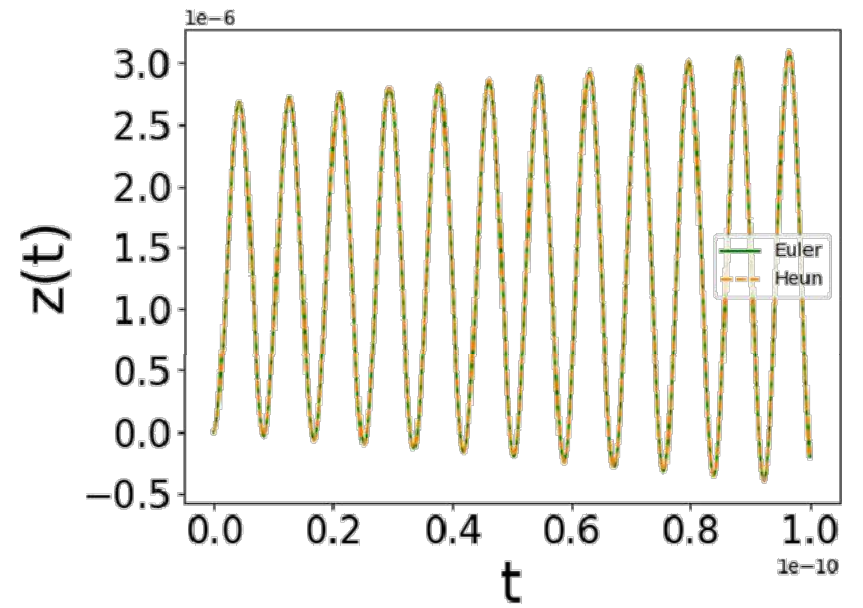
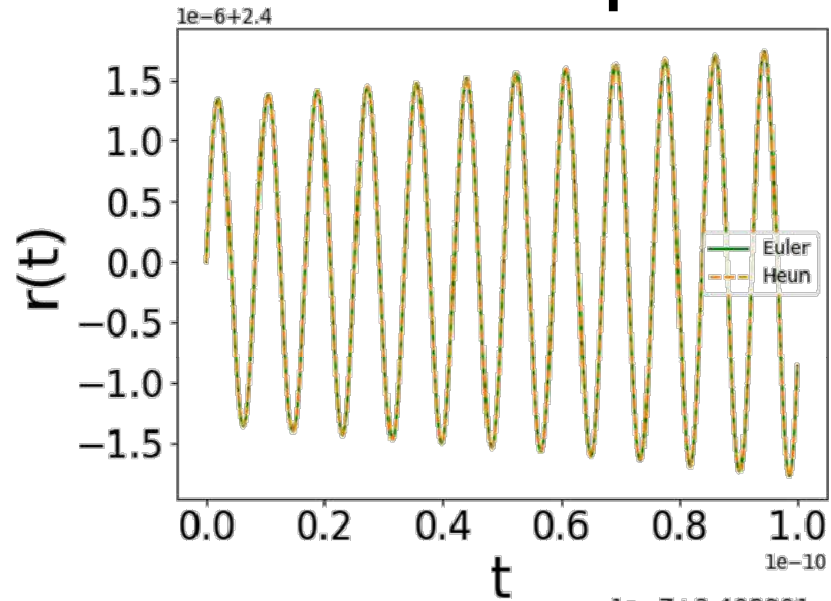
Résultats Obtenus avec la Méthode d'Euler



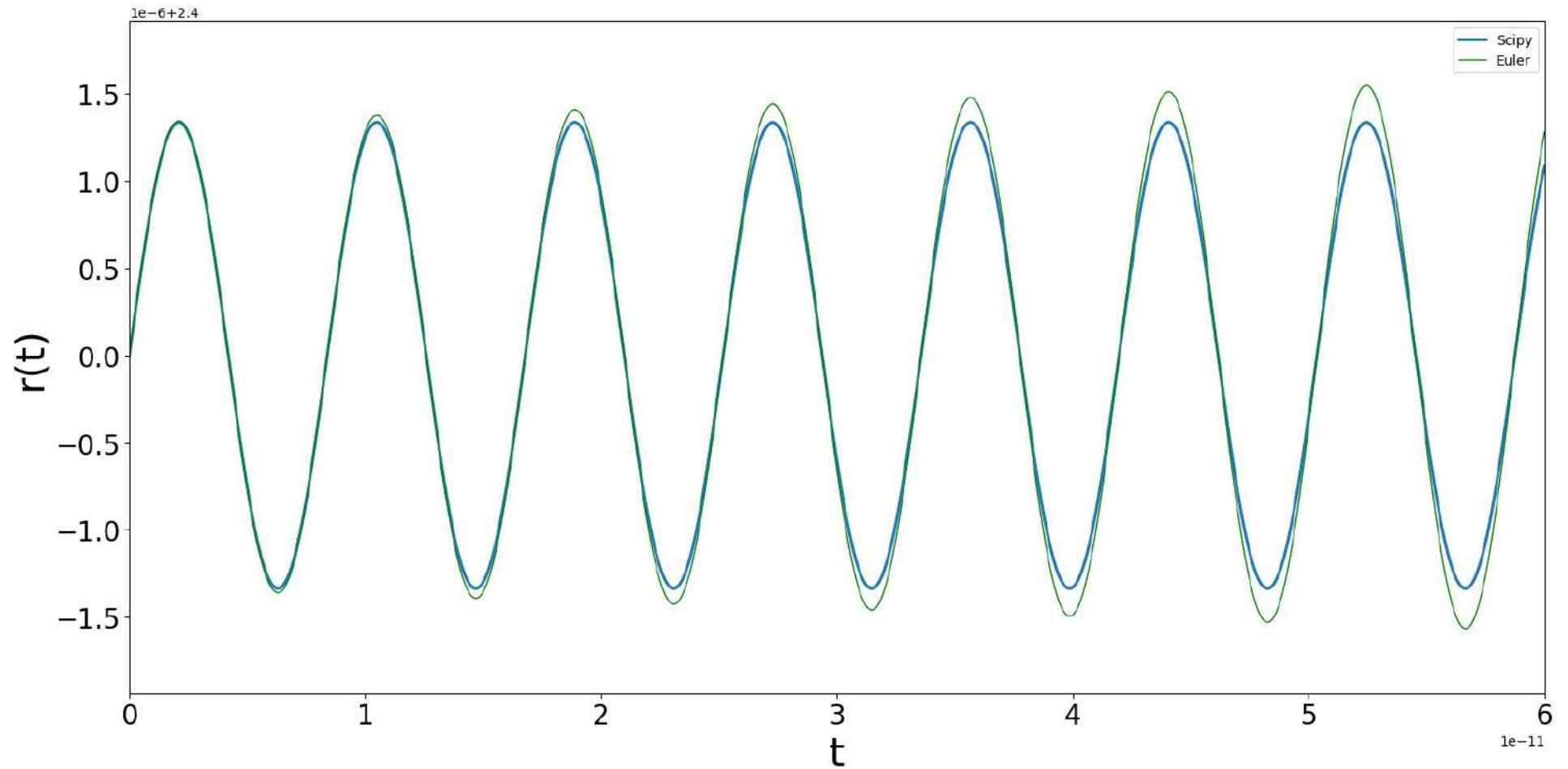
Résultats Obtenus avec la Méthode de Heun



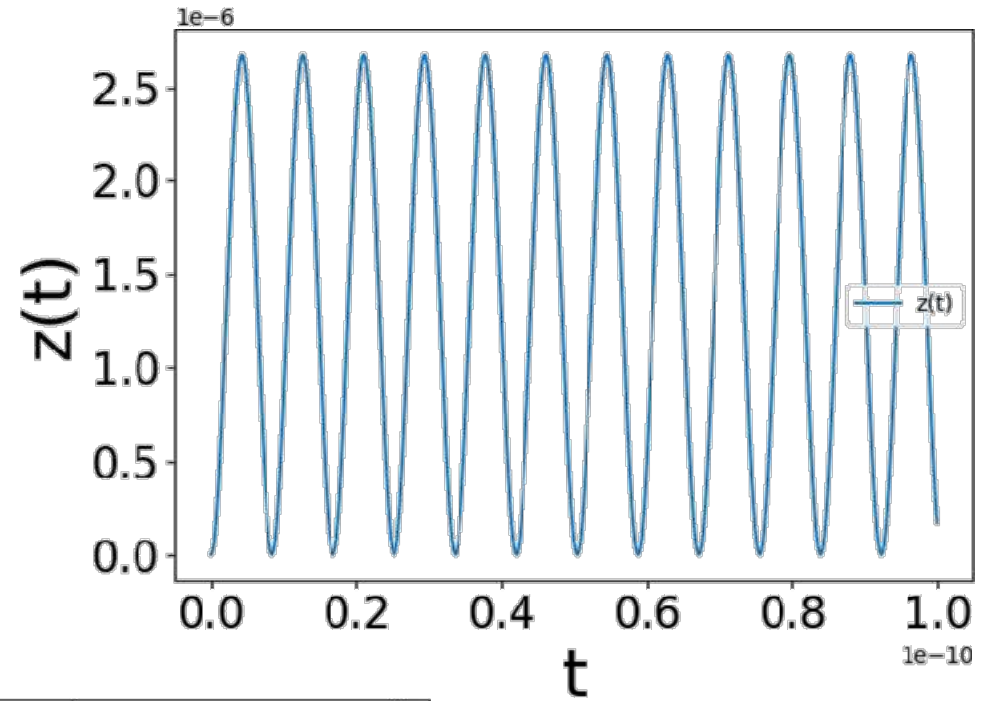
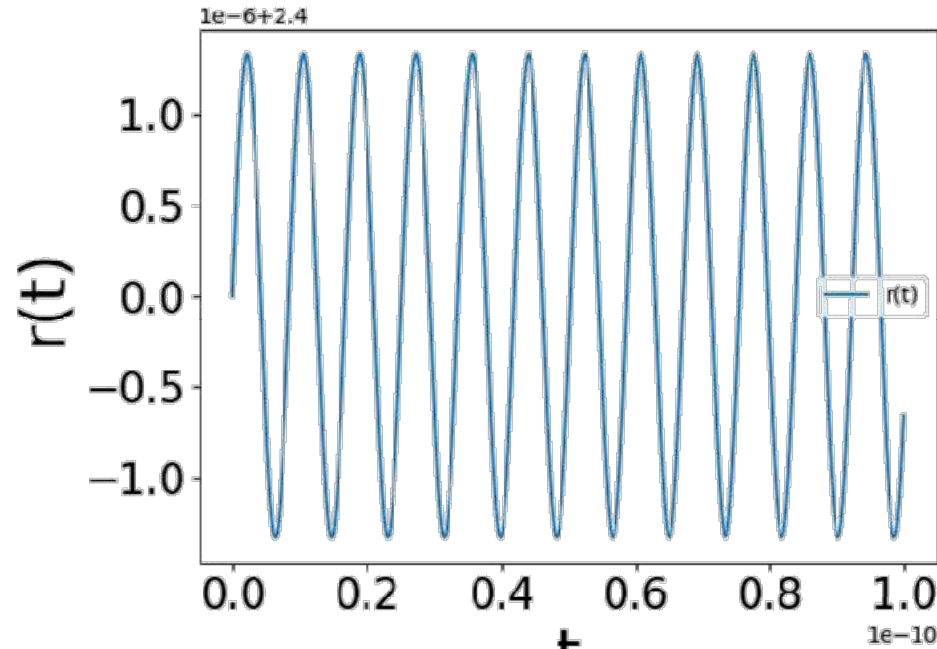
Comparaison des deux Méthodes



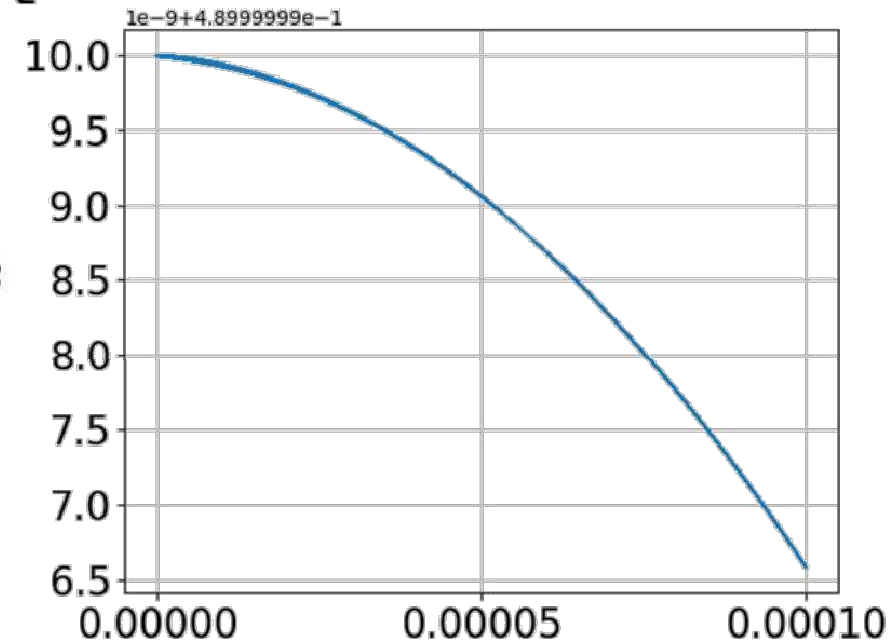
Comparaison de la méthode d'Euler et odeint



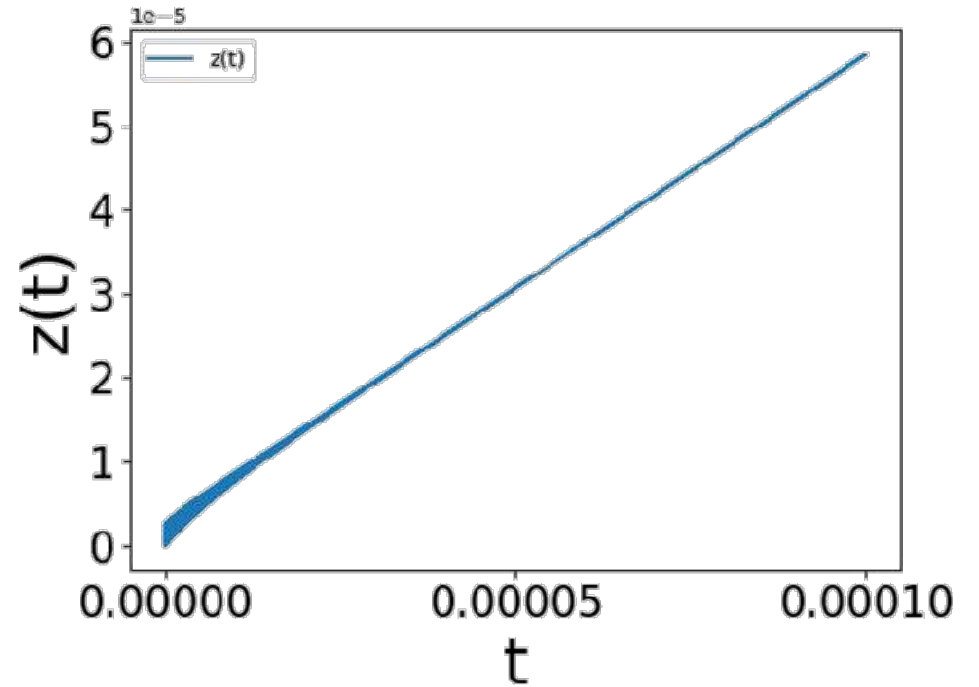
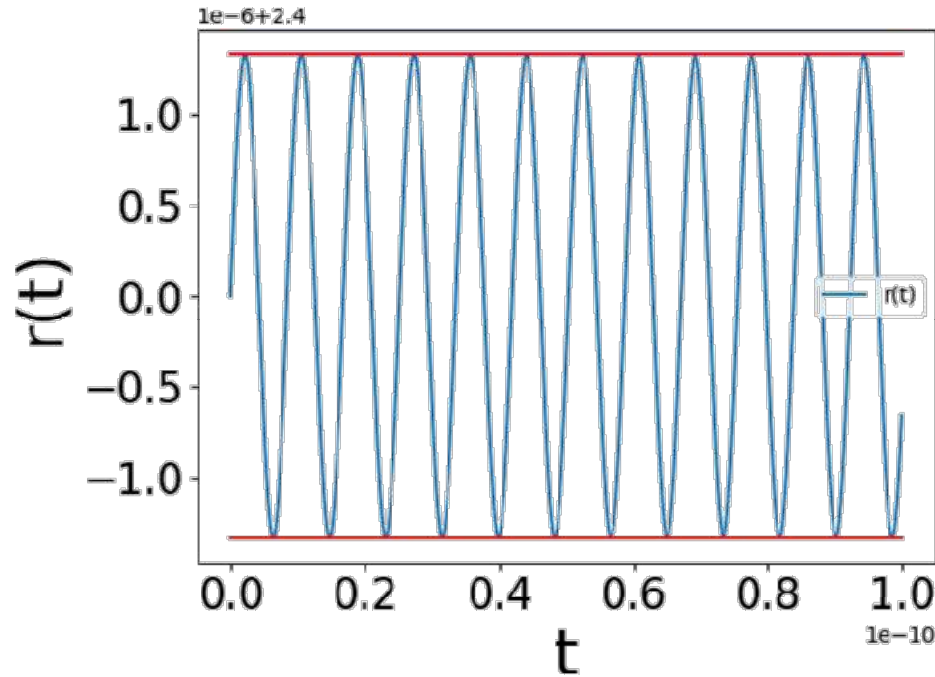
Résultats Obtenus avec le module Scipy



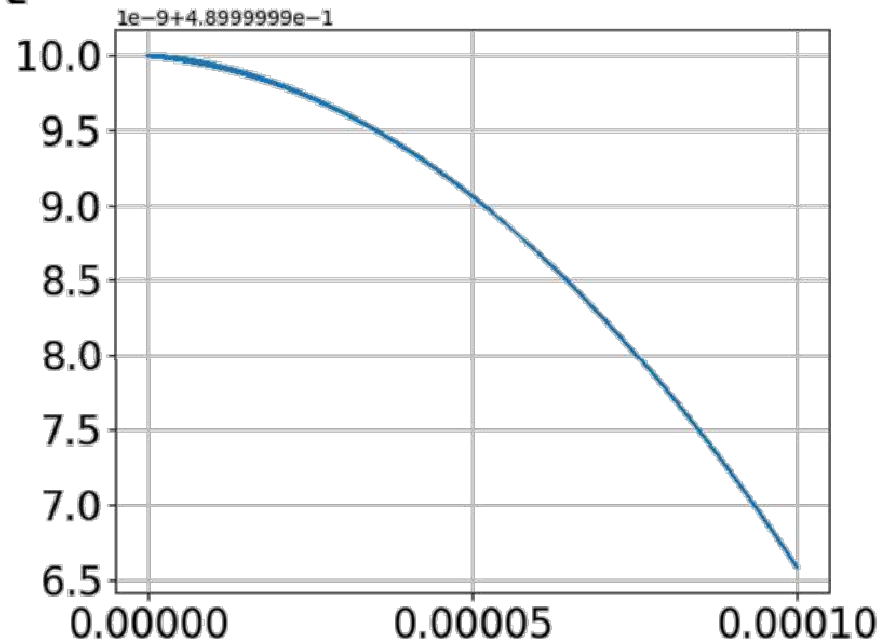
$$0 \leq \rho^2 - (r - R)^2 - z^2$$



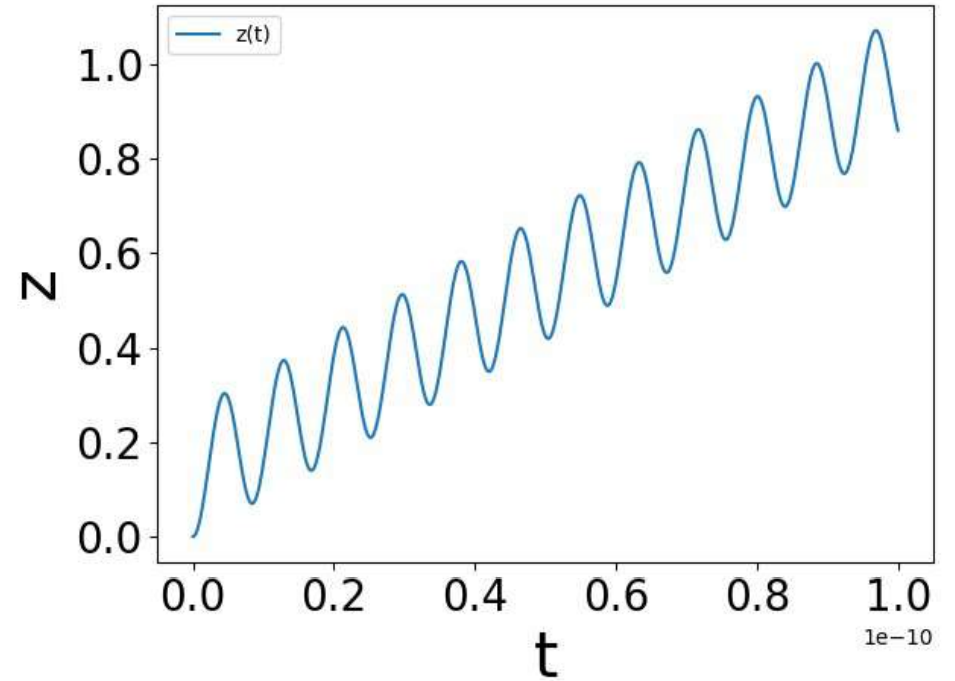
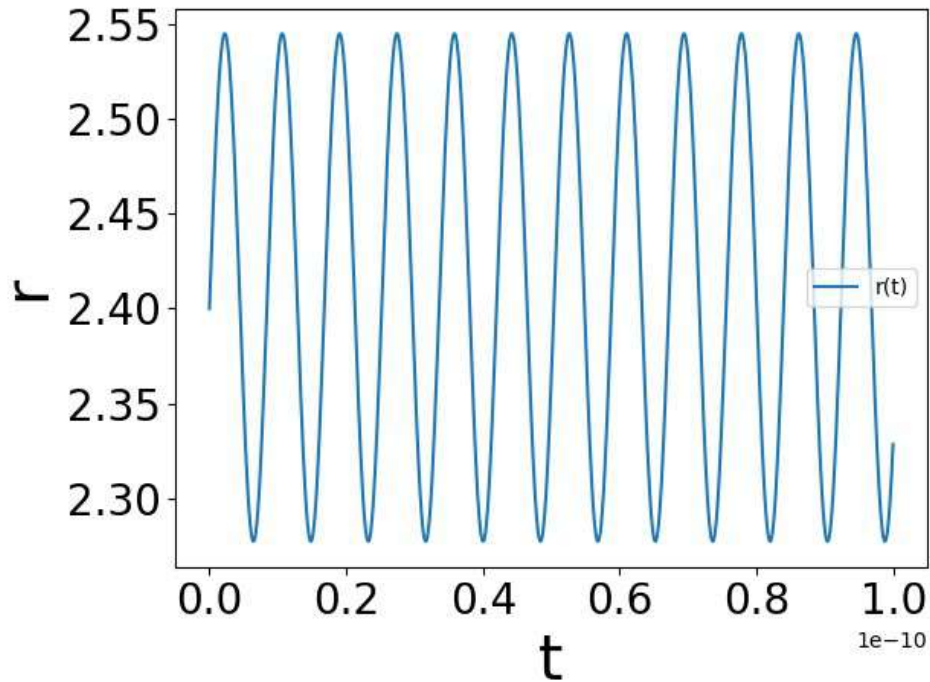
Résultats Obtenus avec le module Scipy



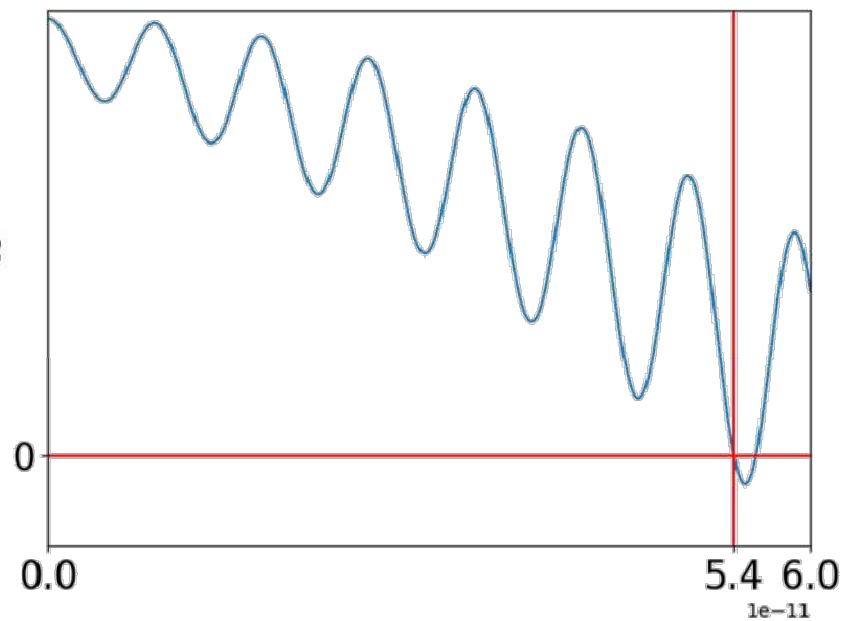
$$0 \leq \rho^2 - (r - R)^2 - z^2$$



Résultats Obtenus avec le module Scipy

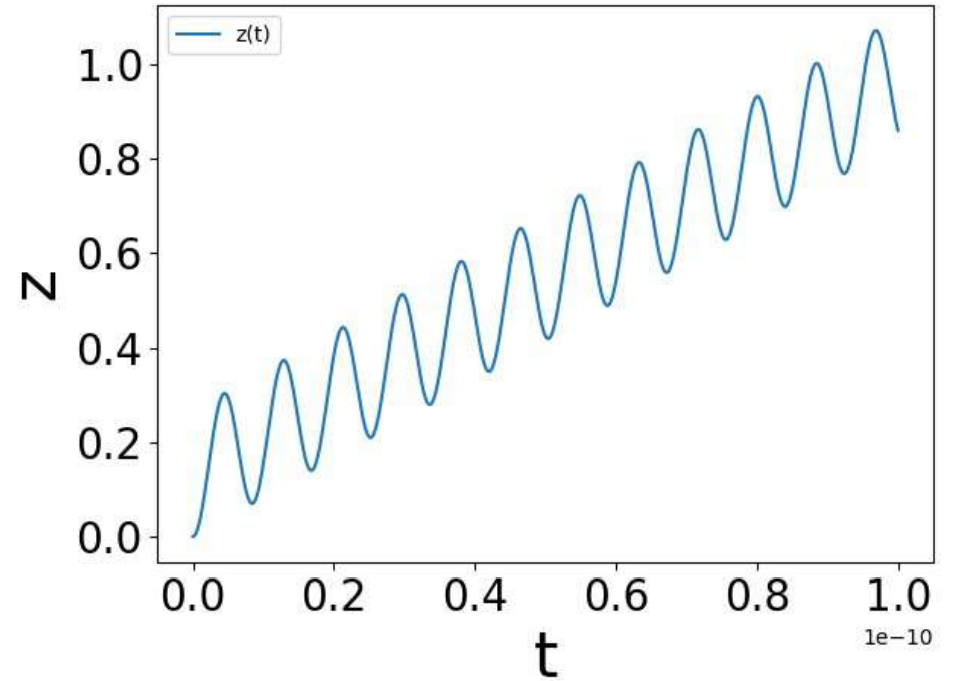
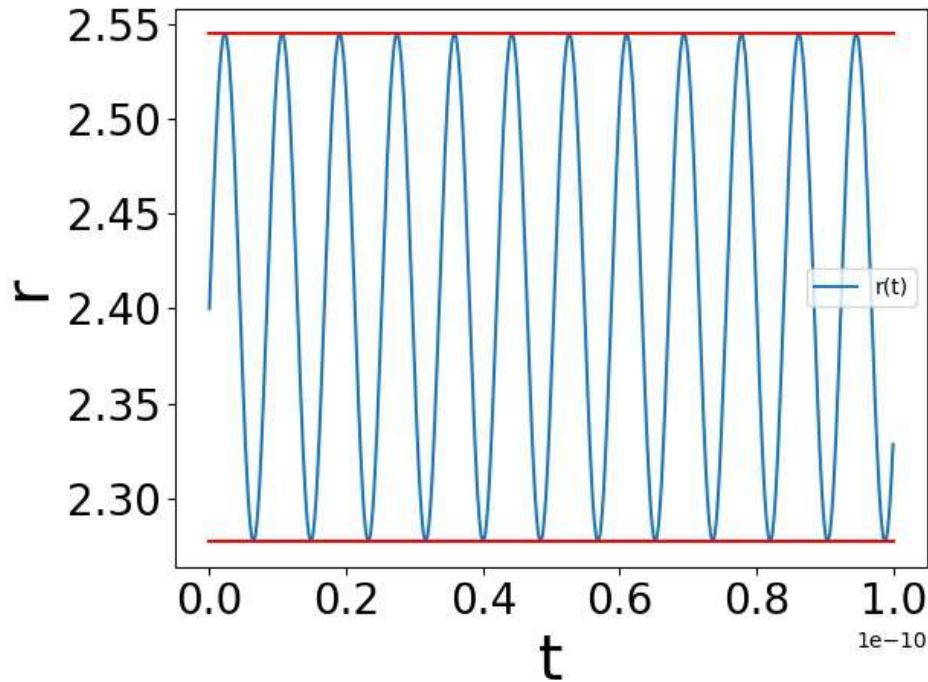


$$0 \leq \rho^2 - (r - R)^2 - z^2$$

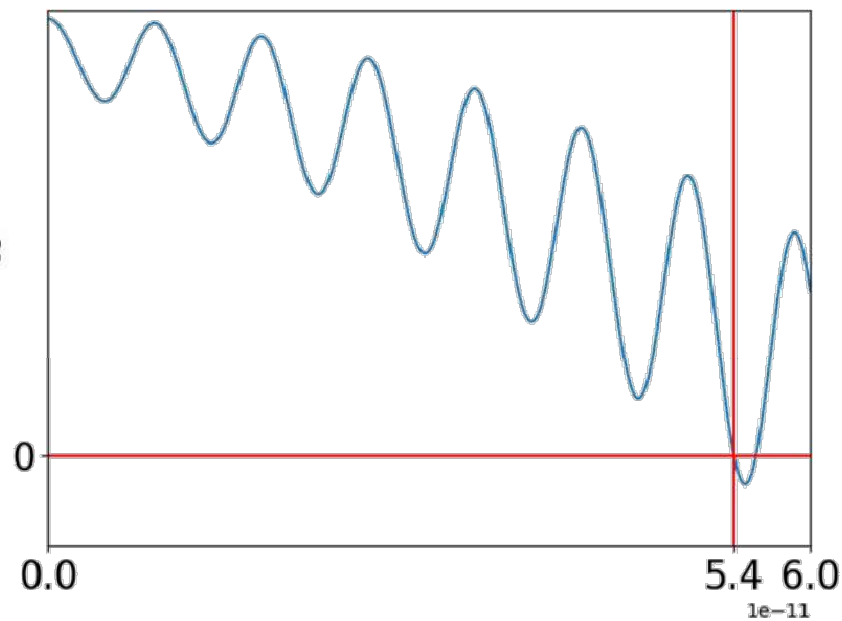


Pour une vitesse initiale plus élevée

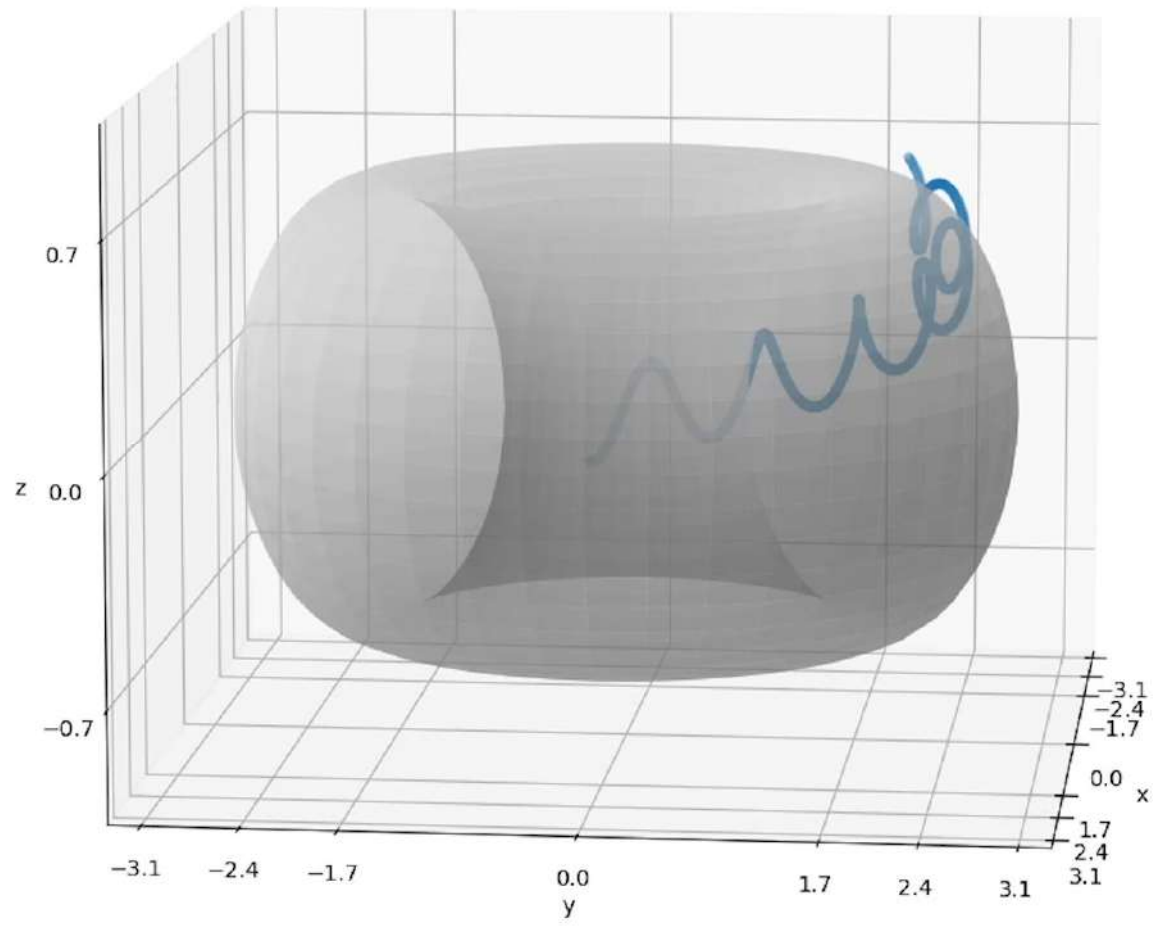
Résultats Obtenus avec le module Scipy

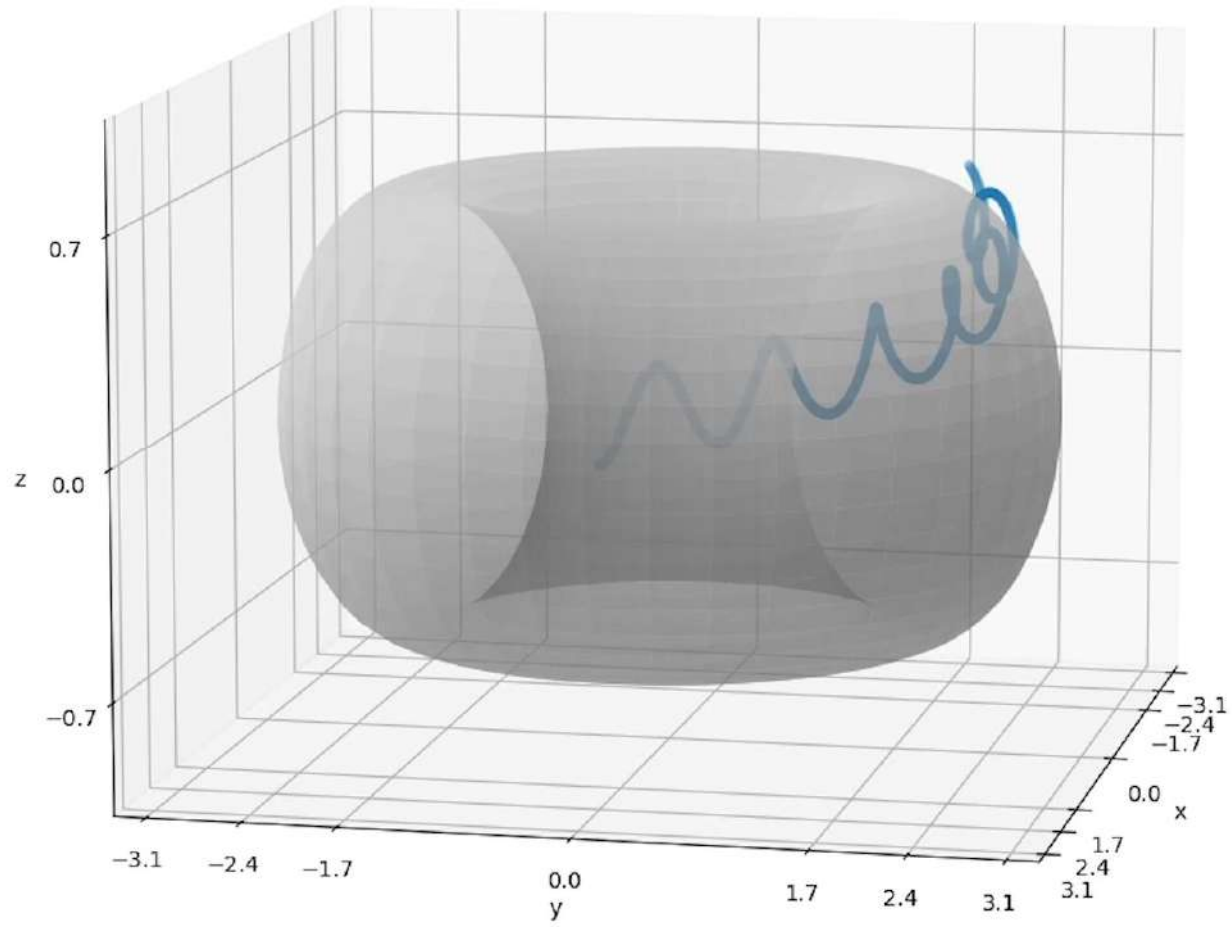


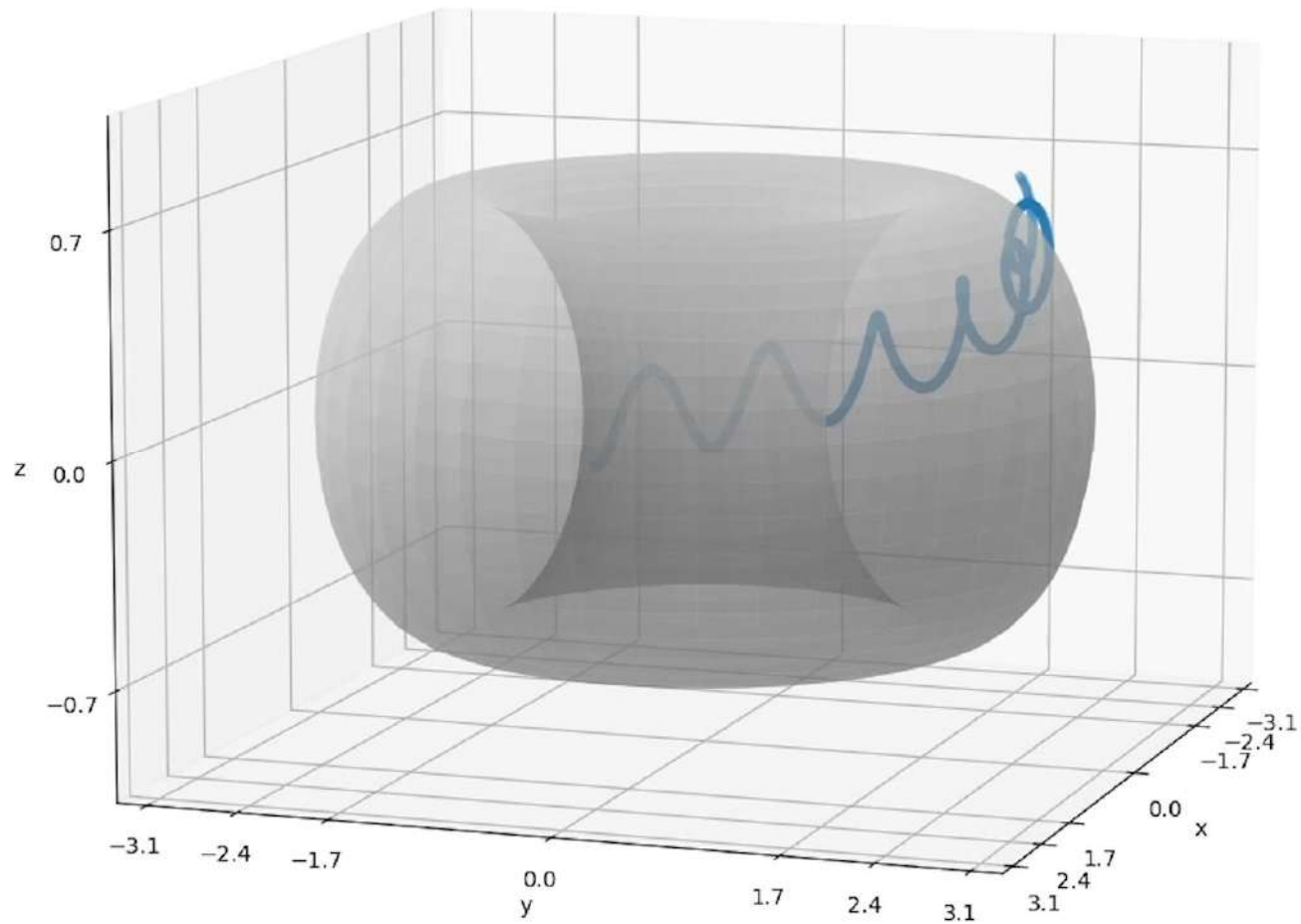
$$0 \leq \rho^2 - (r - R)^2 - z^2$$

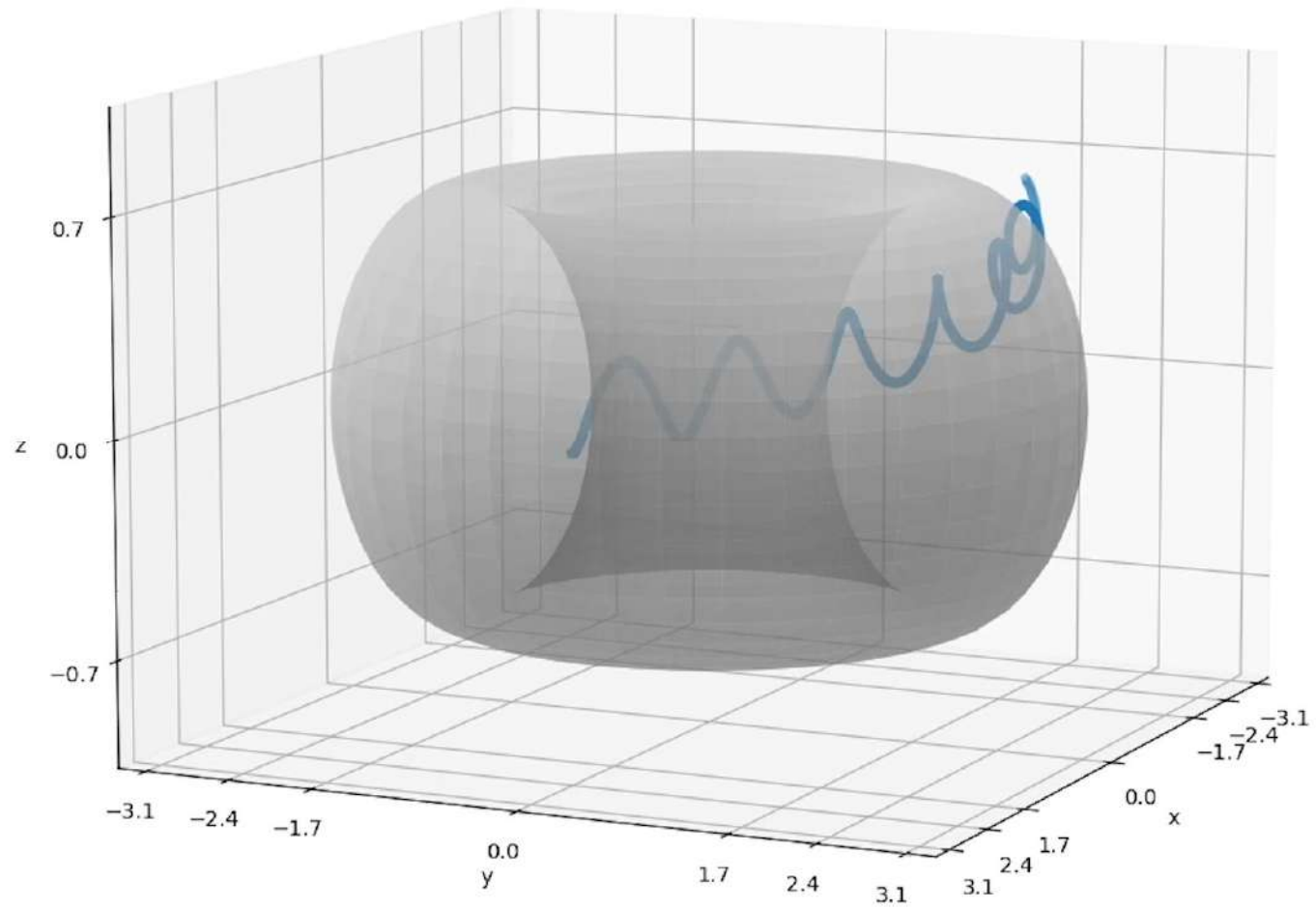


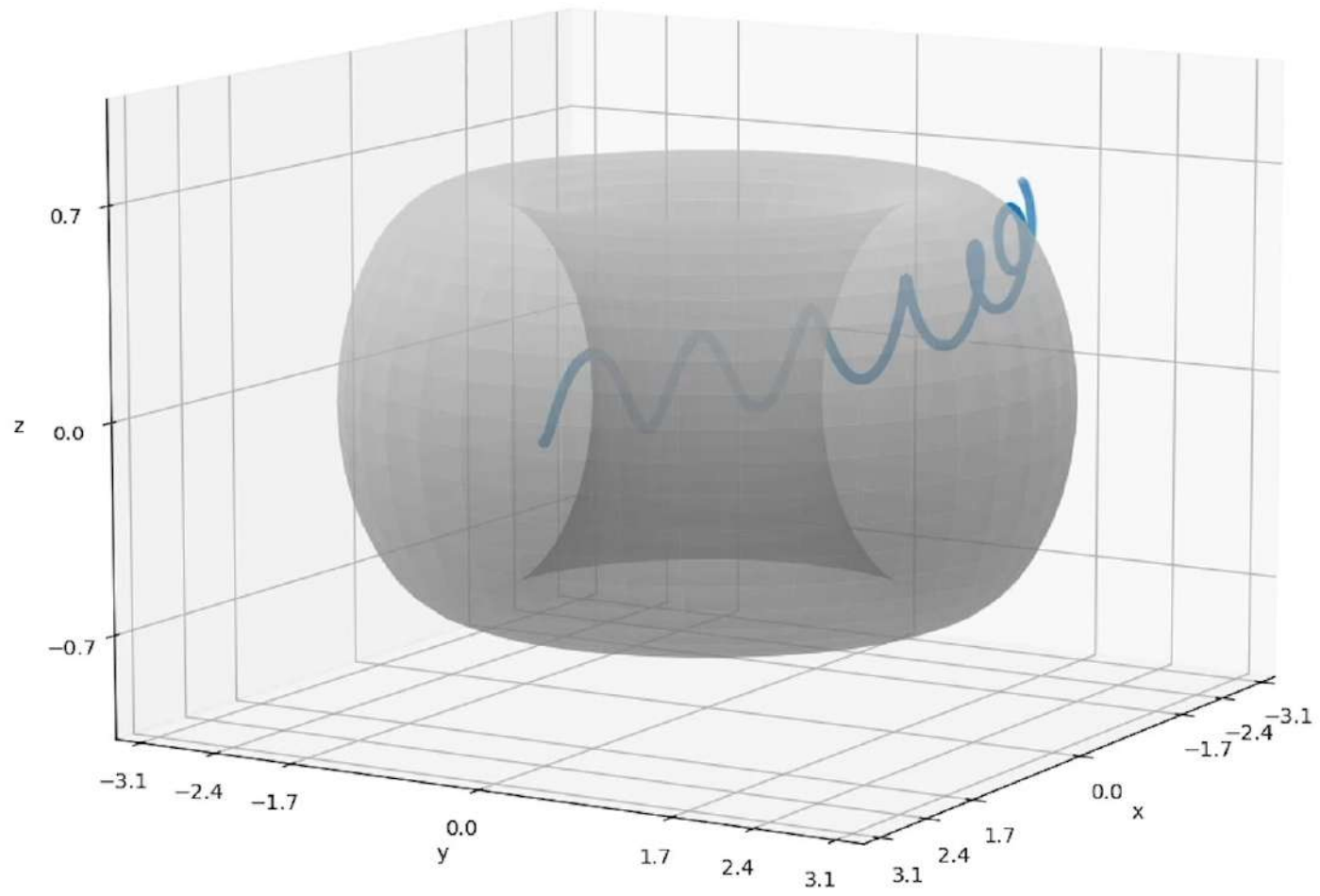
Pour une vitesse initiale plus élevée

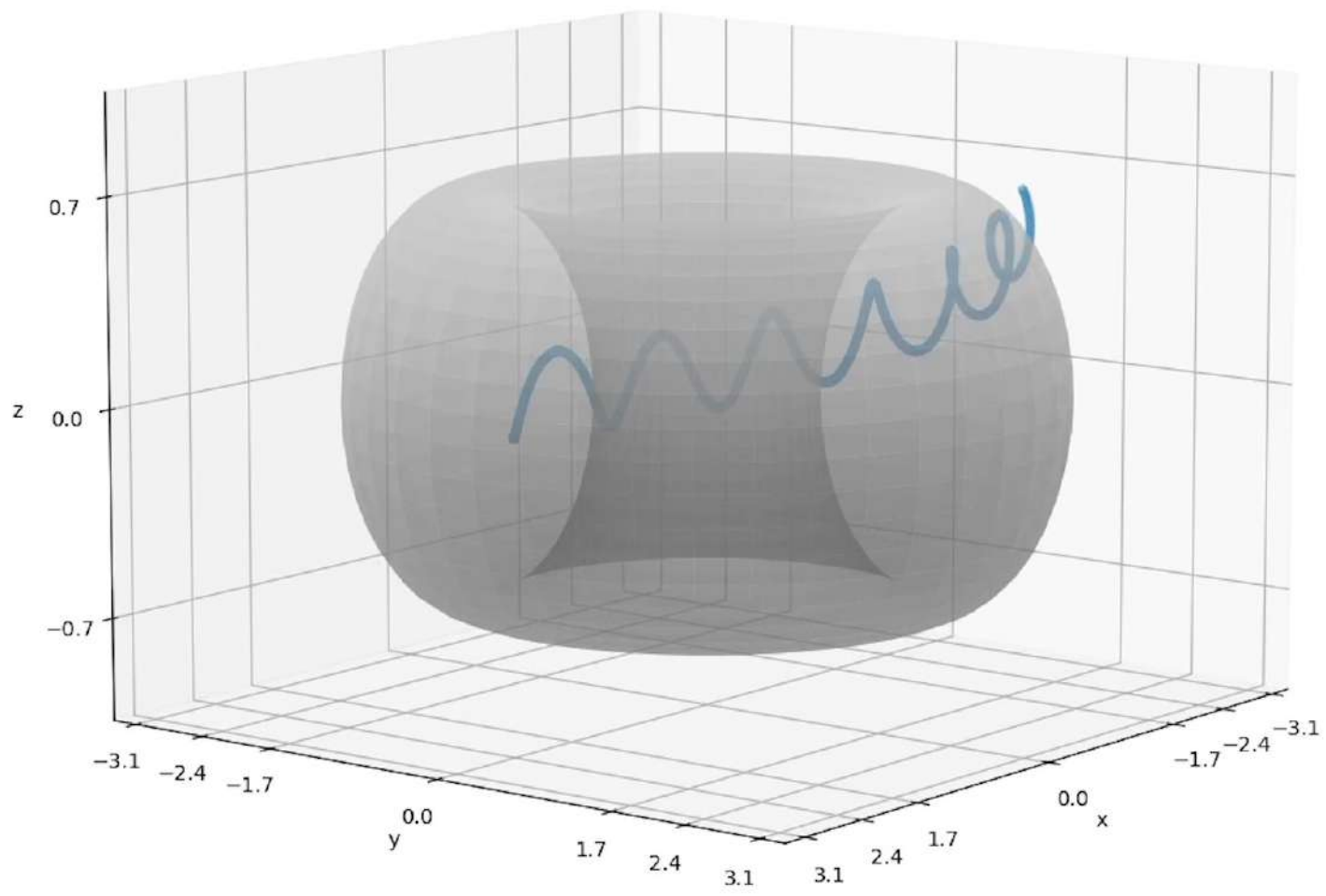


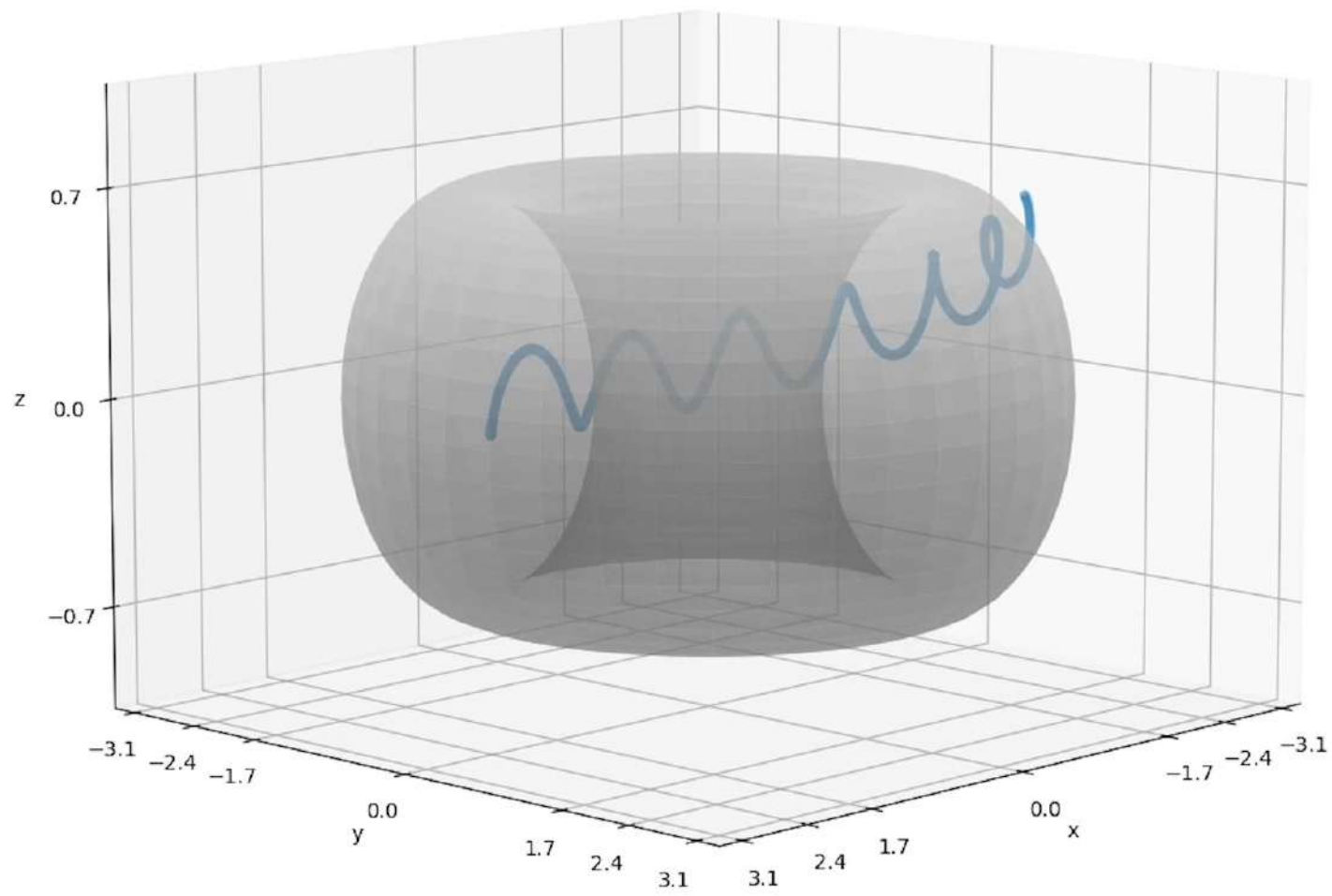


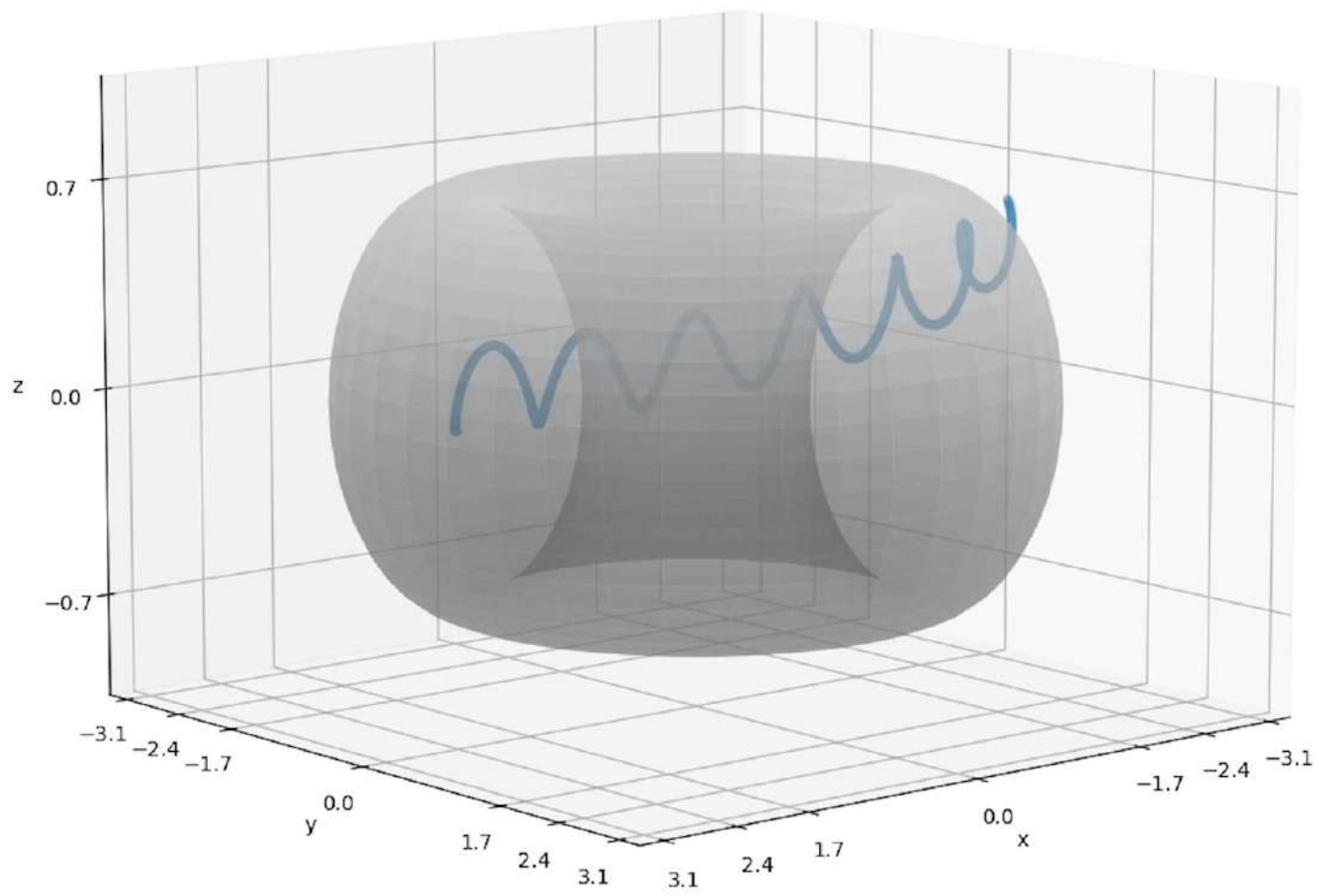


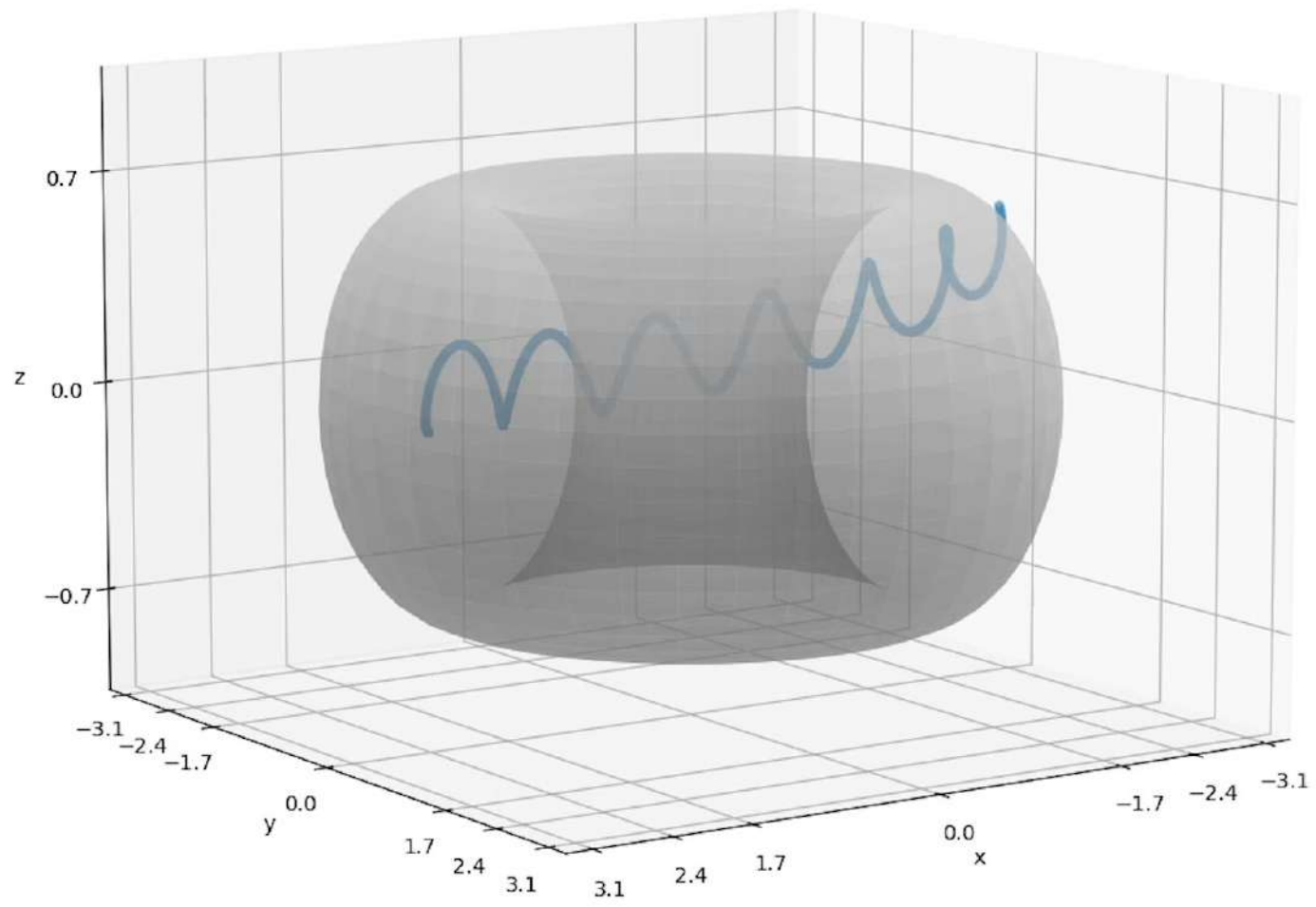


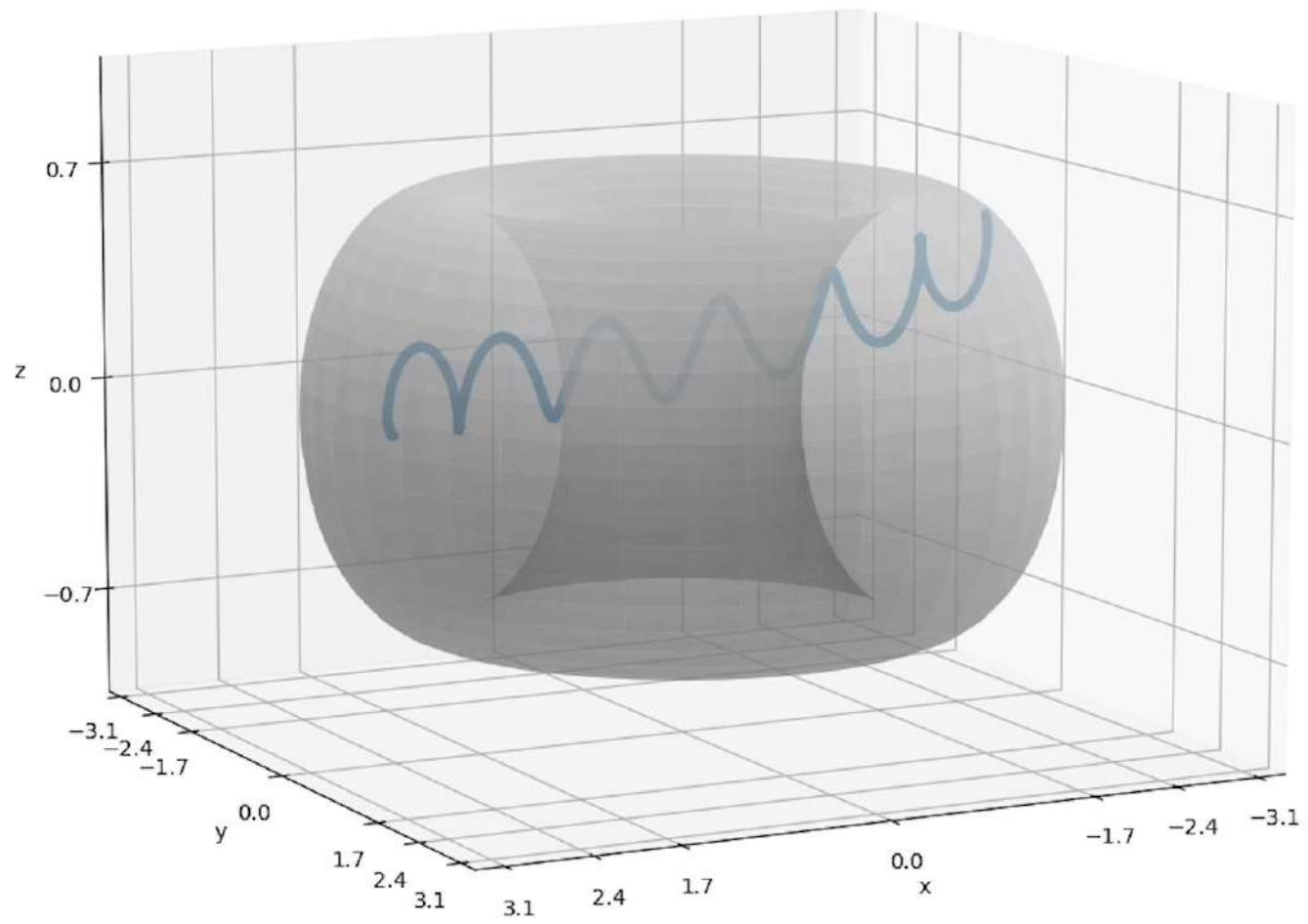


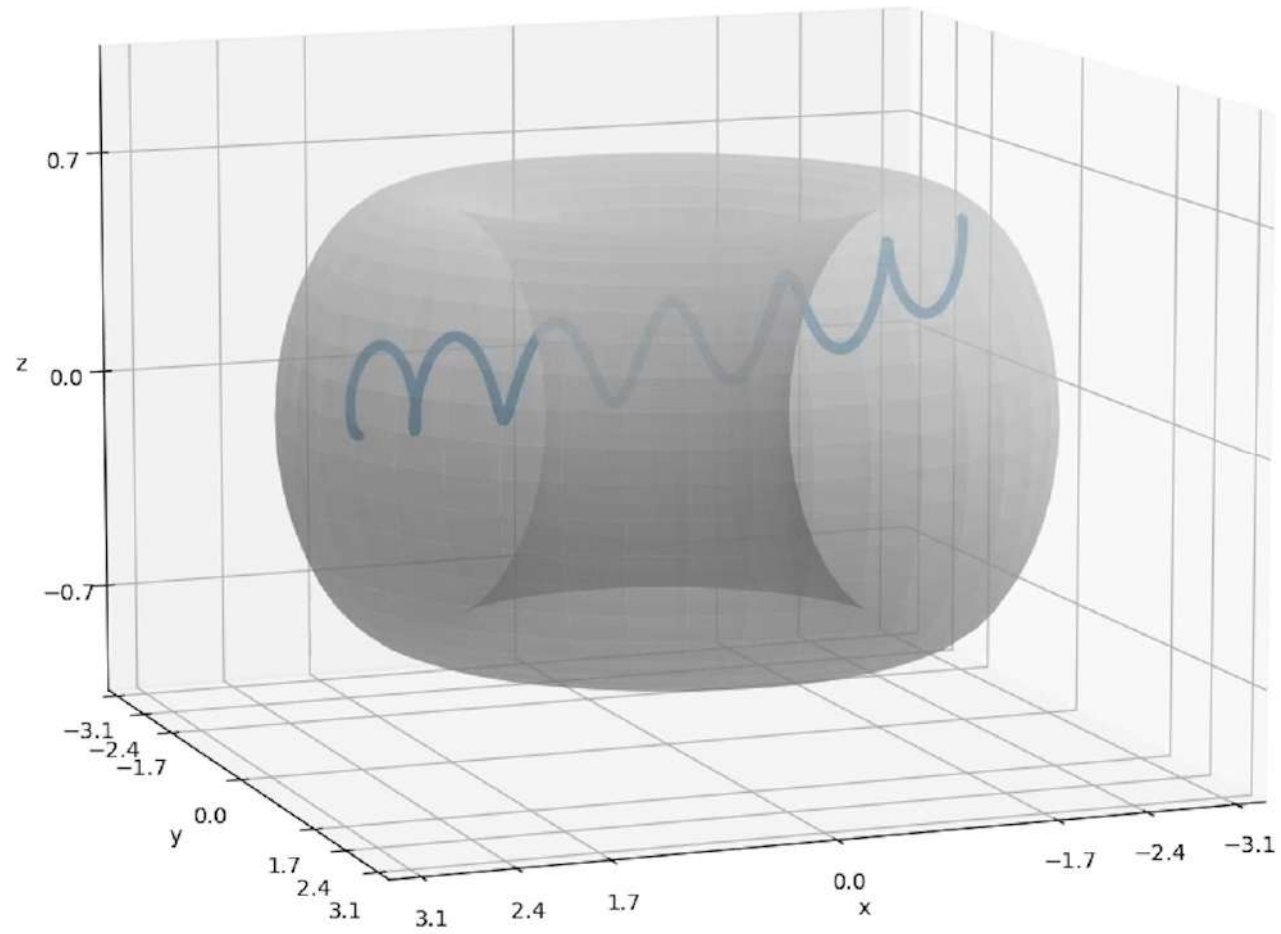


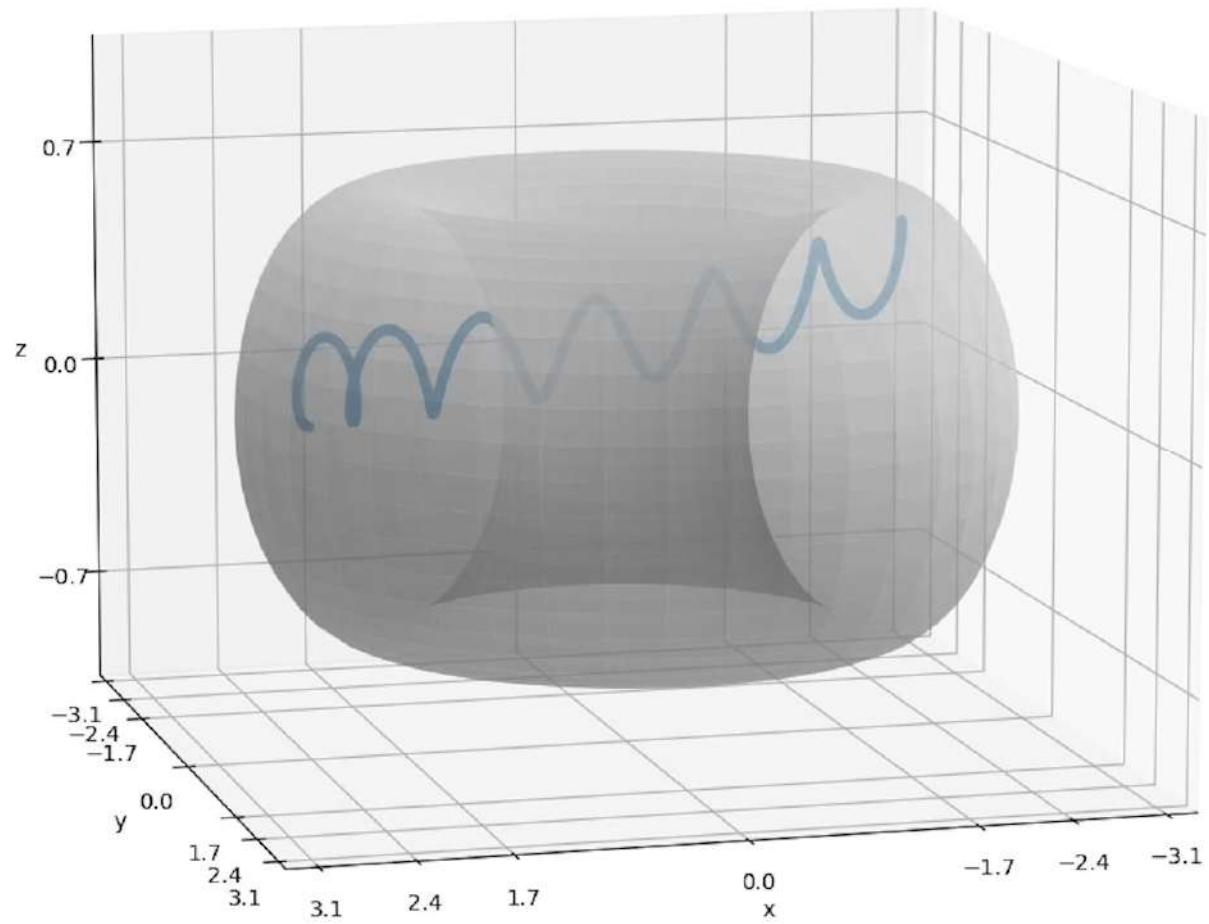


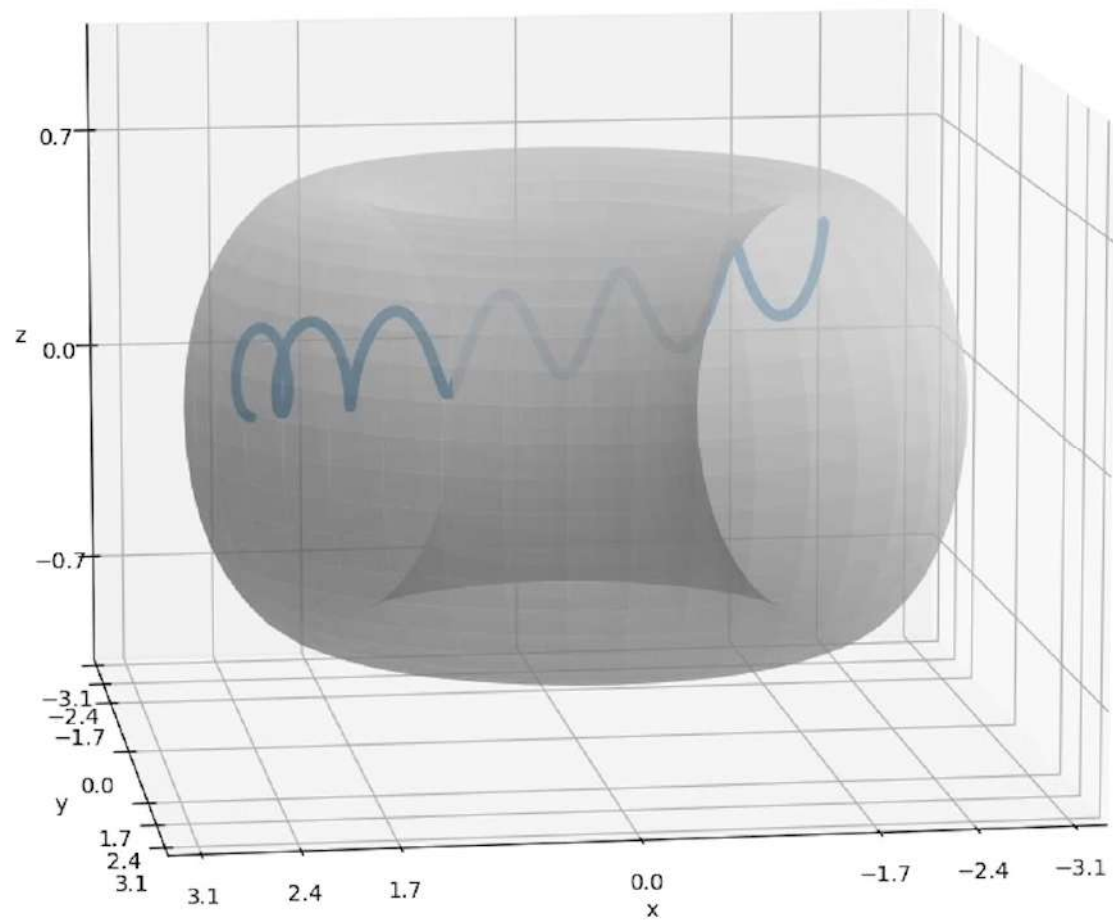


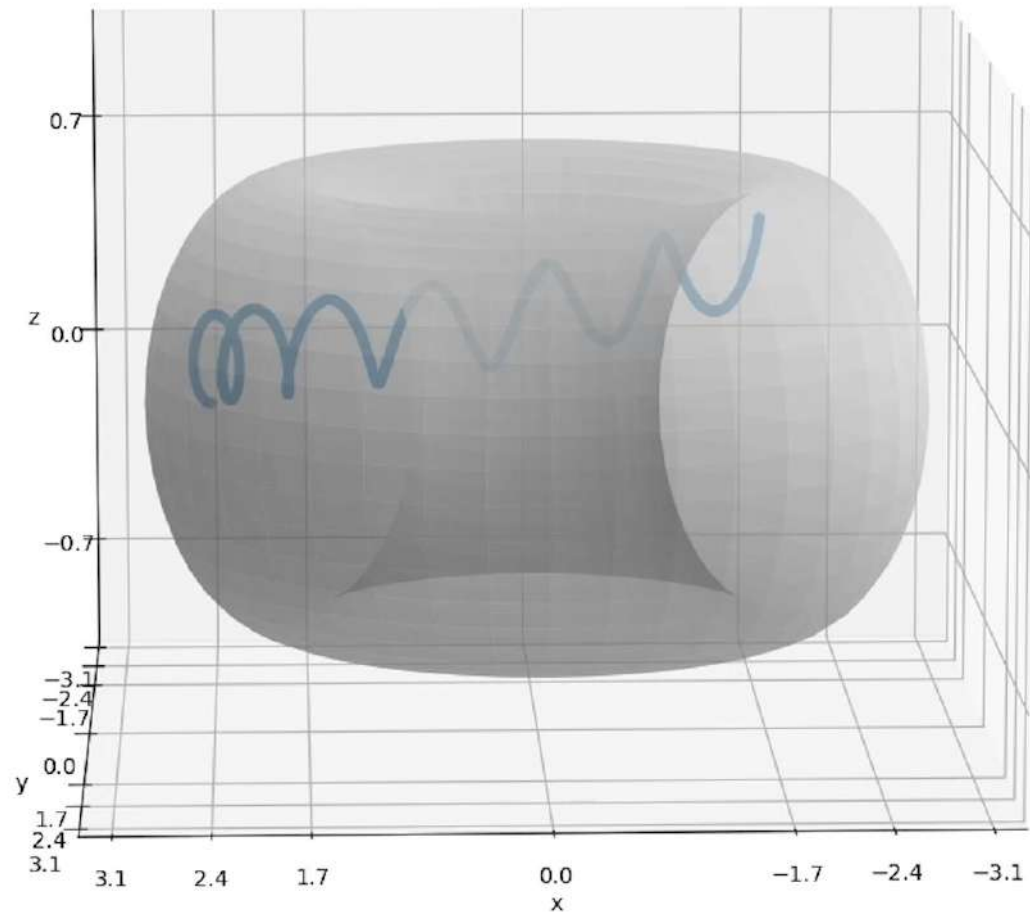


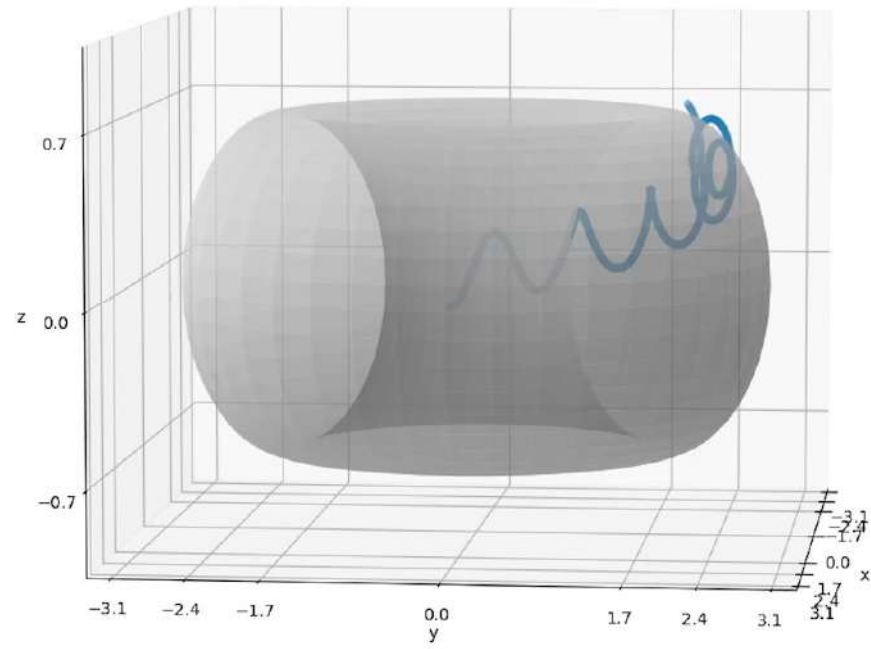


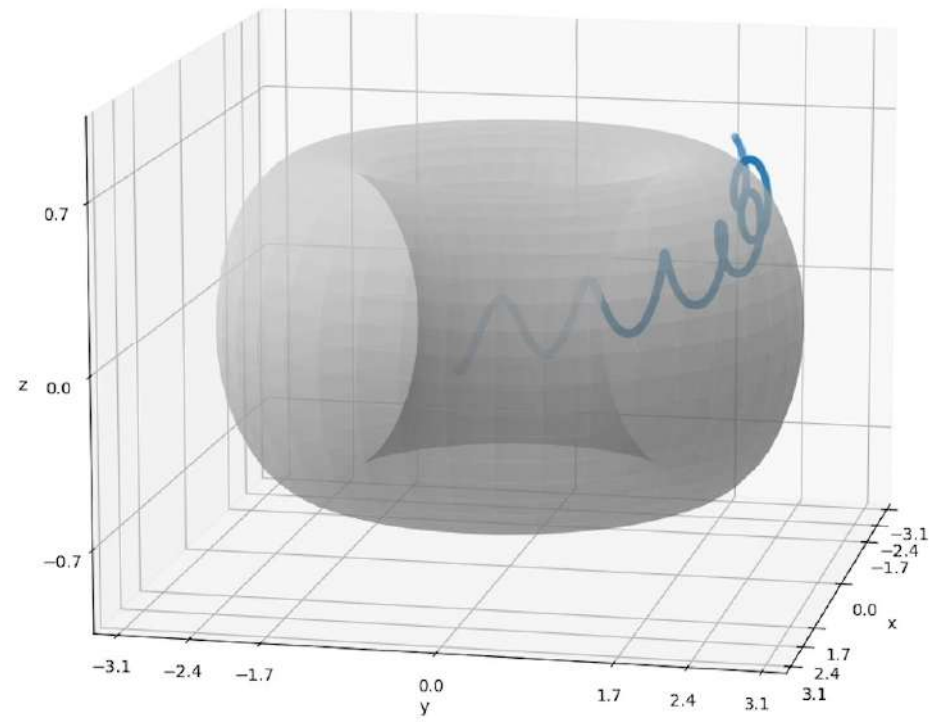


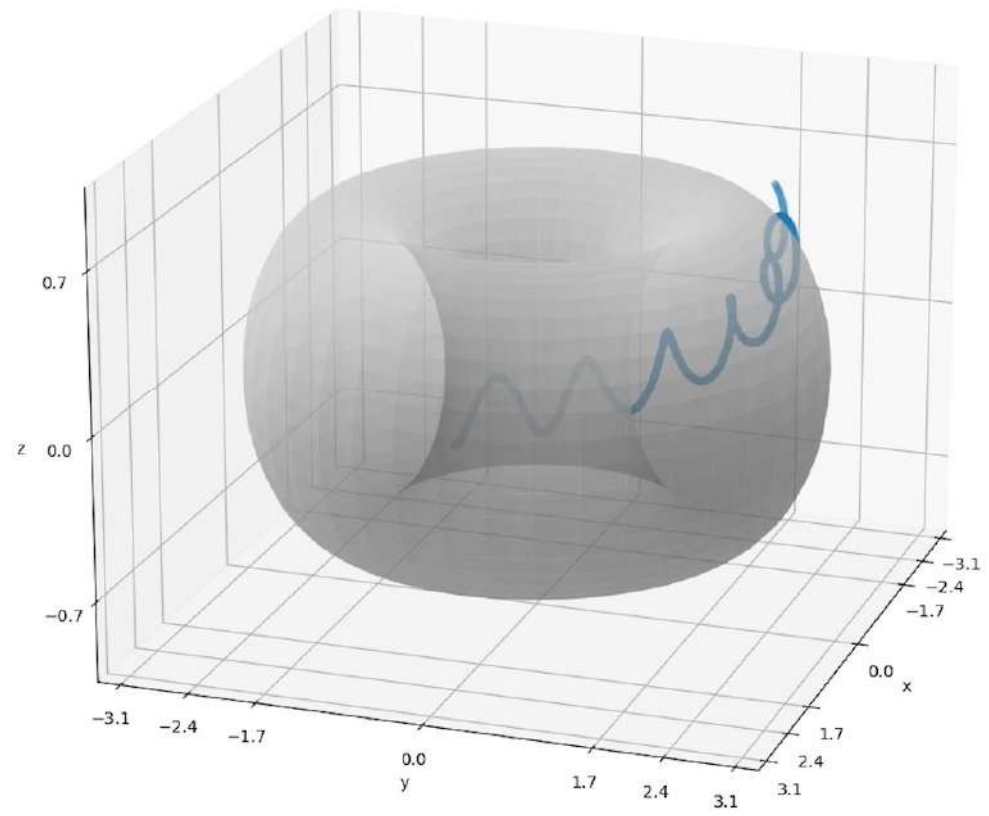


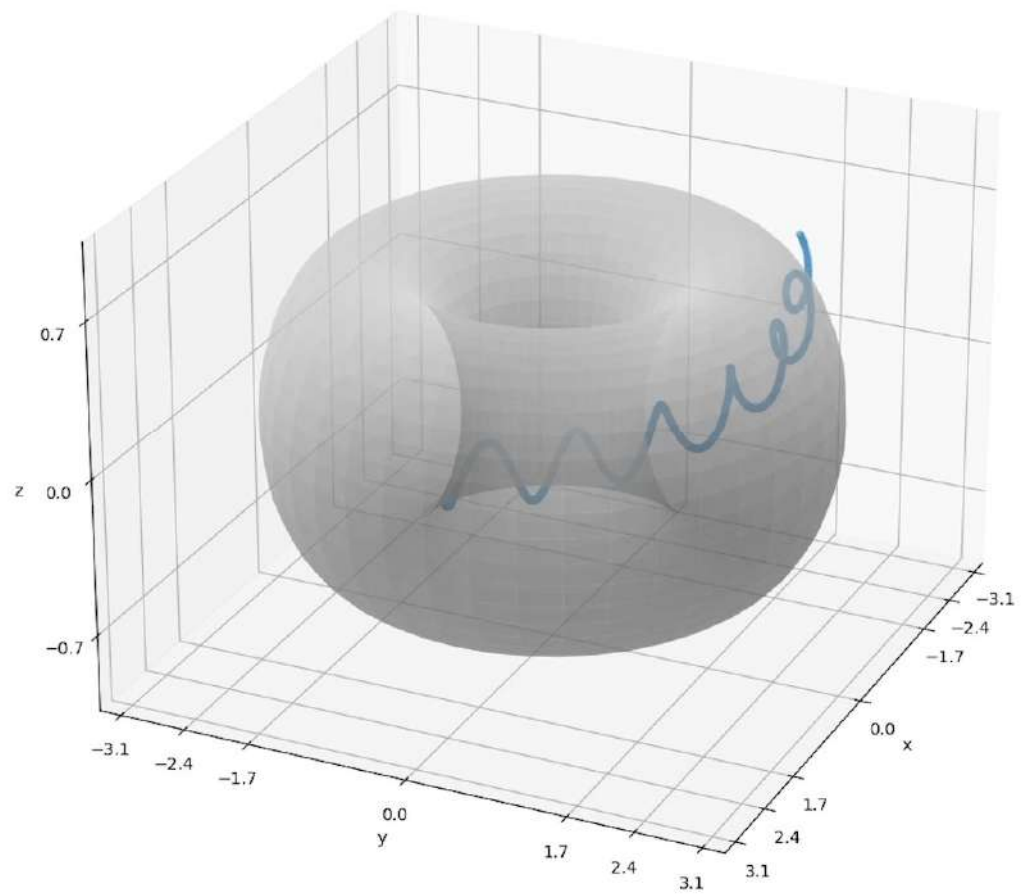


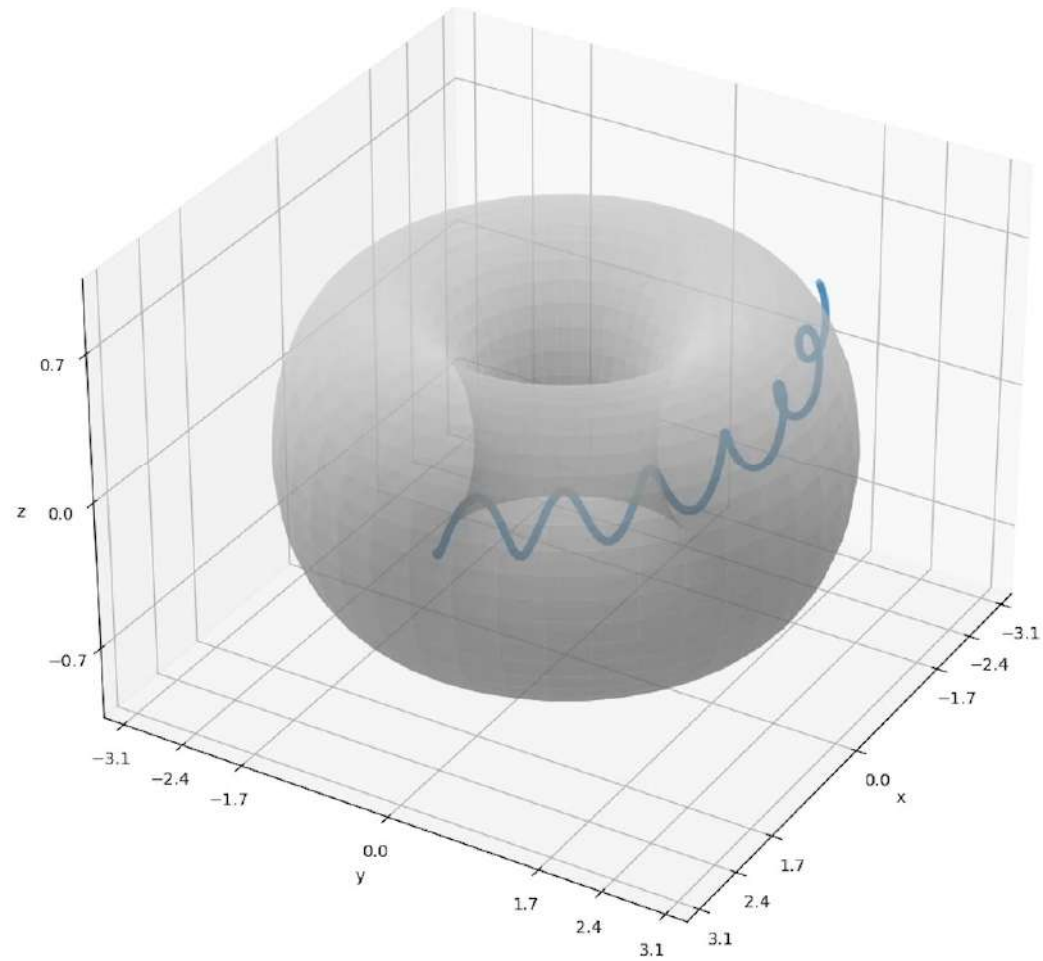


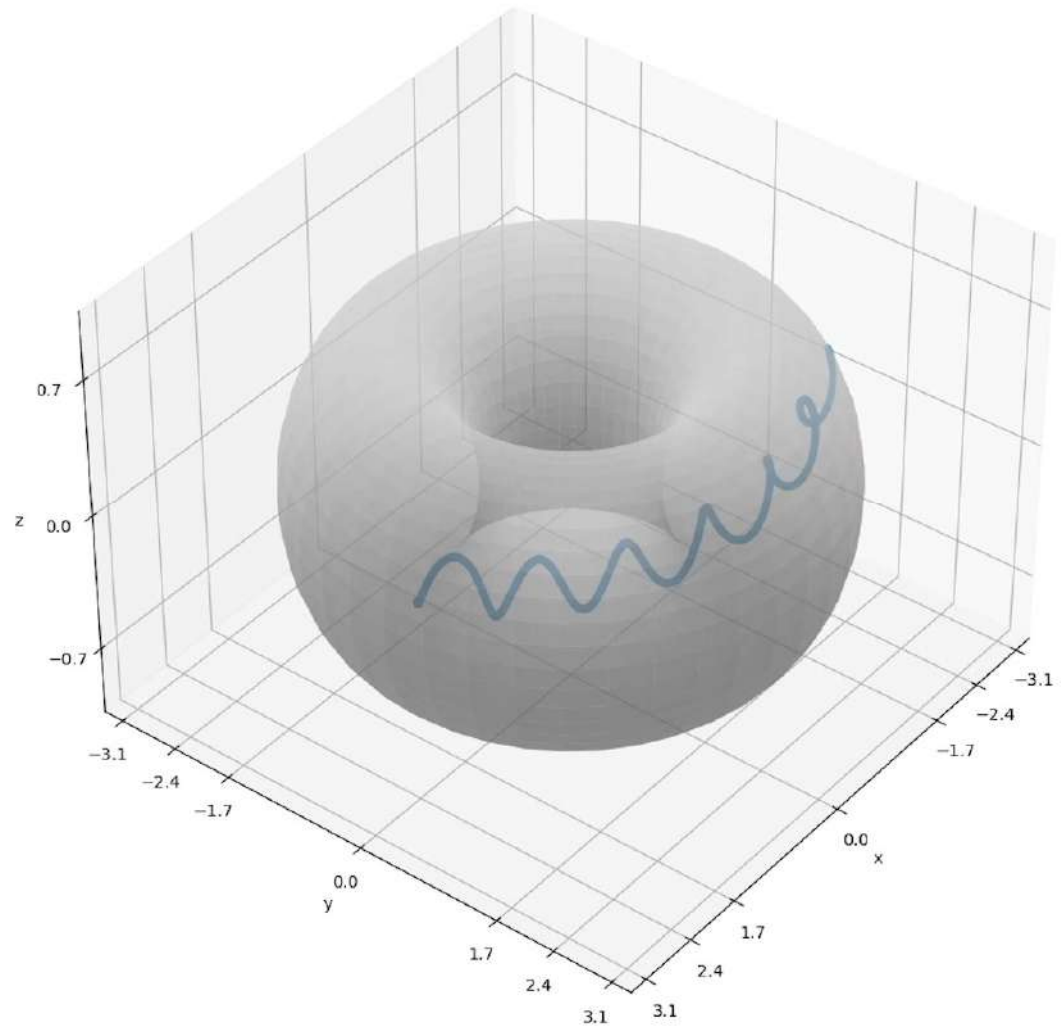


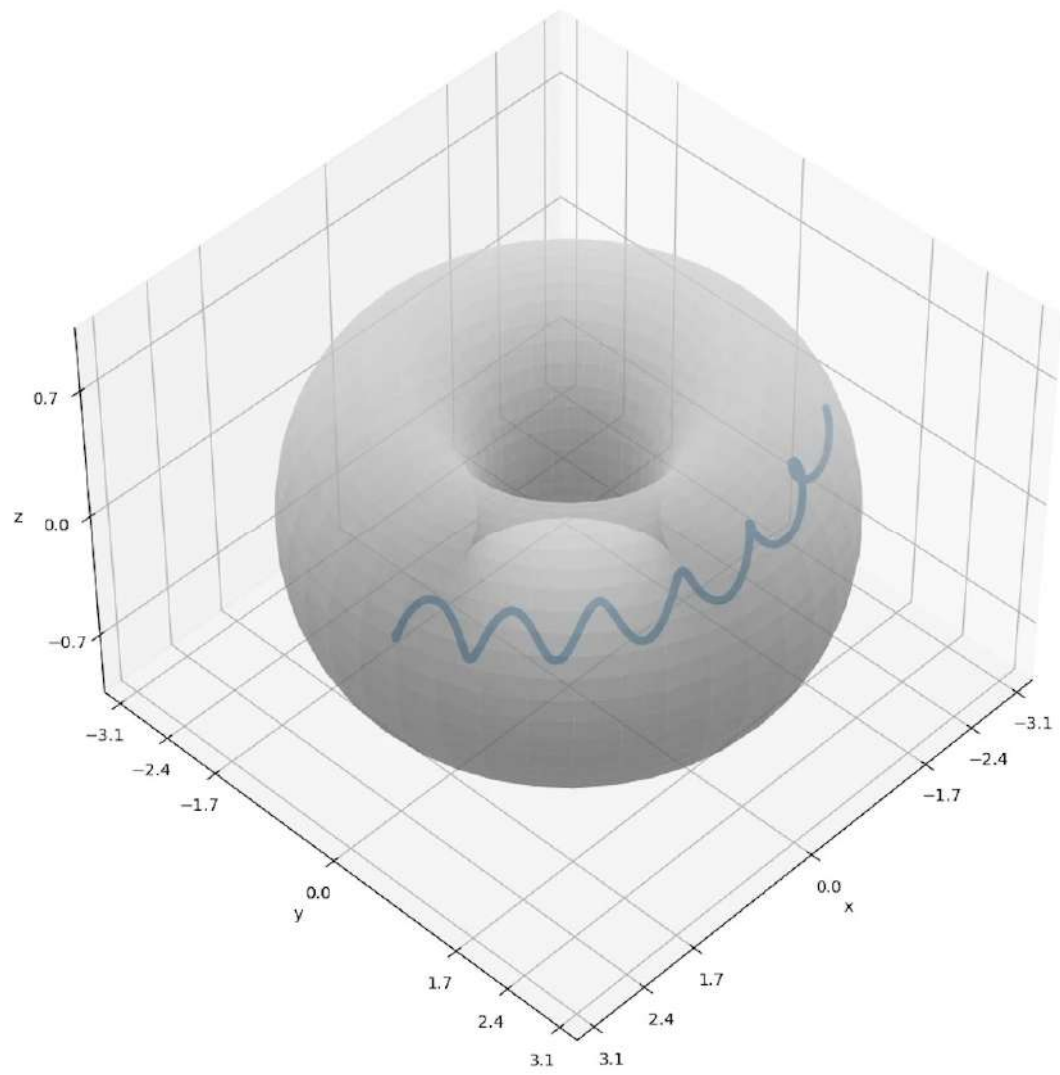


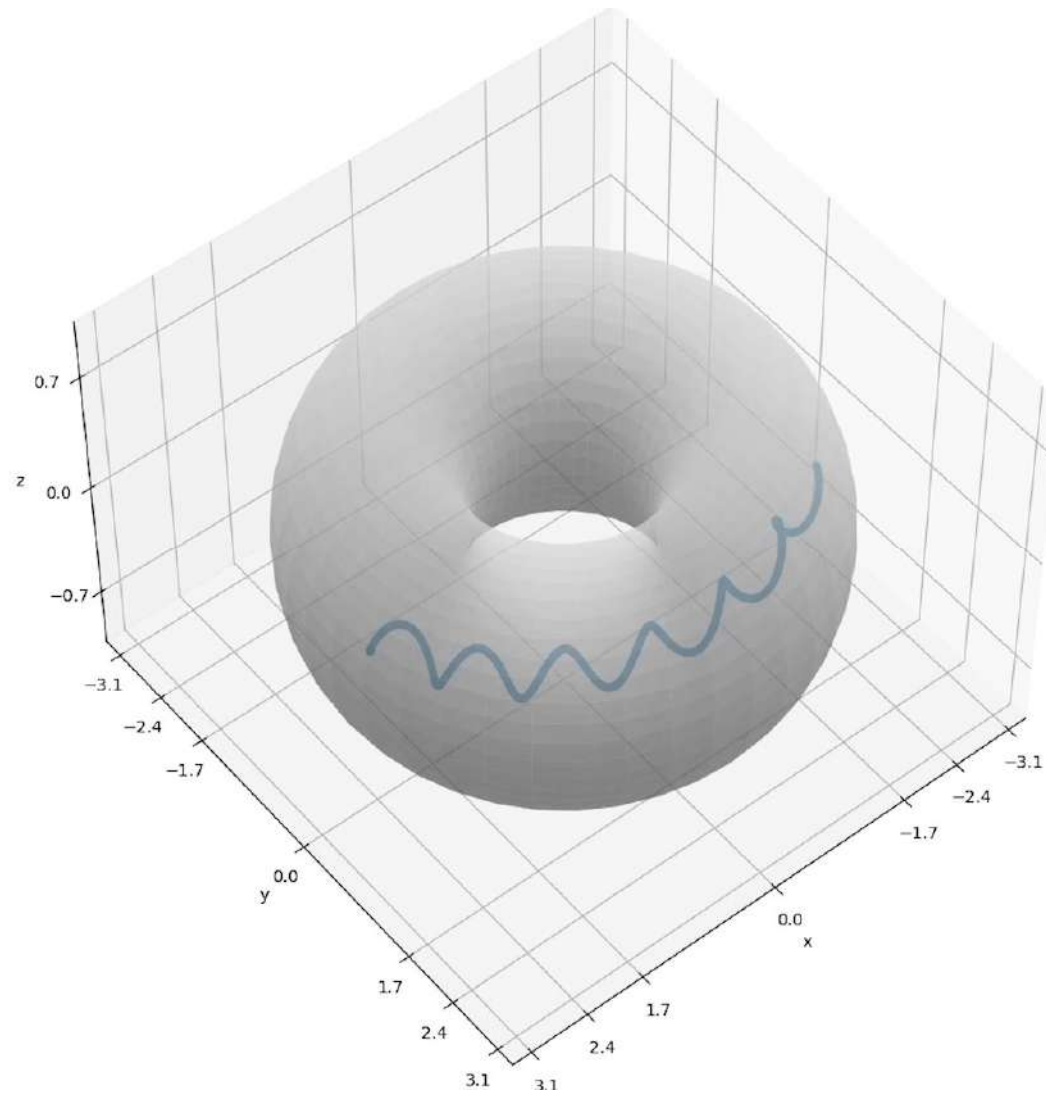


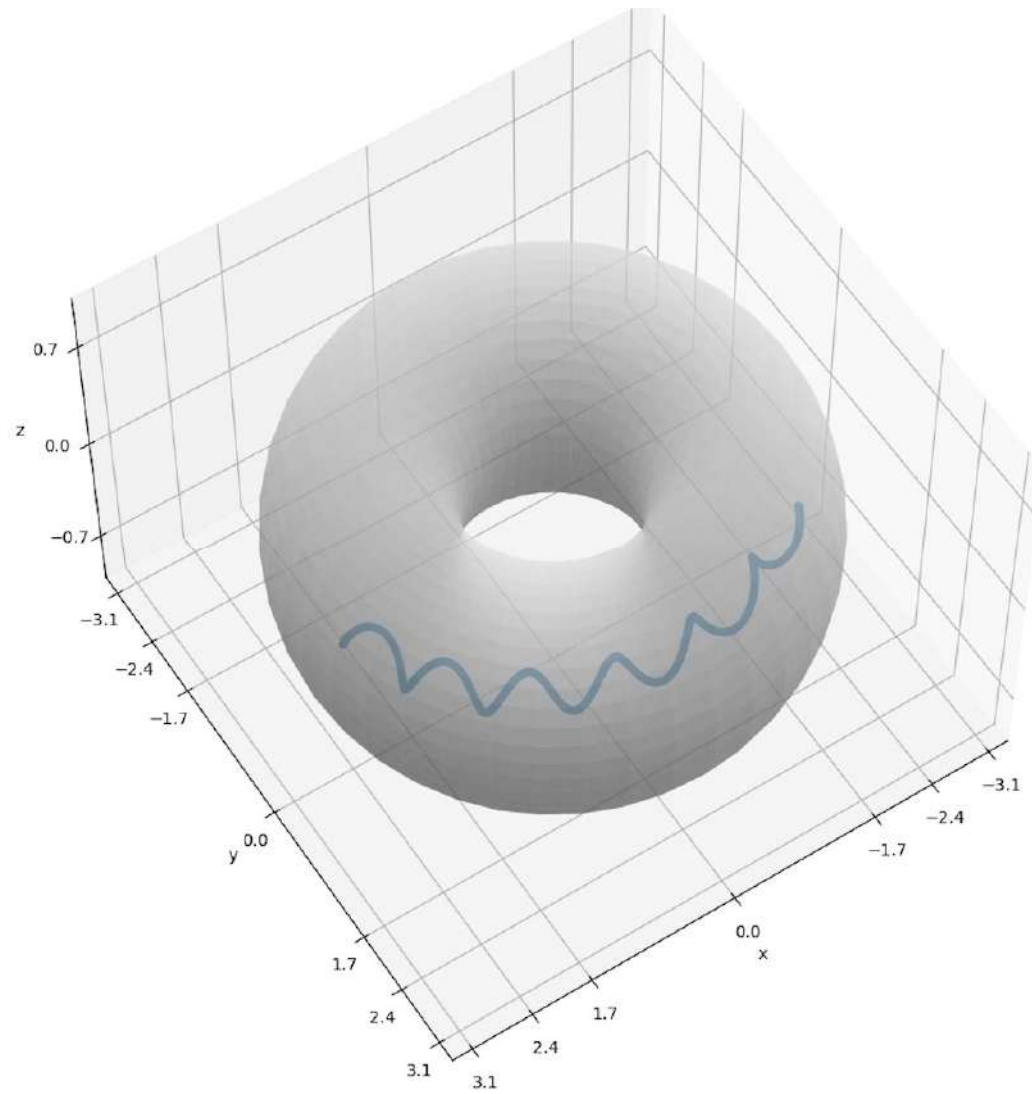


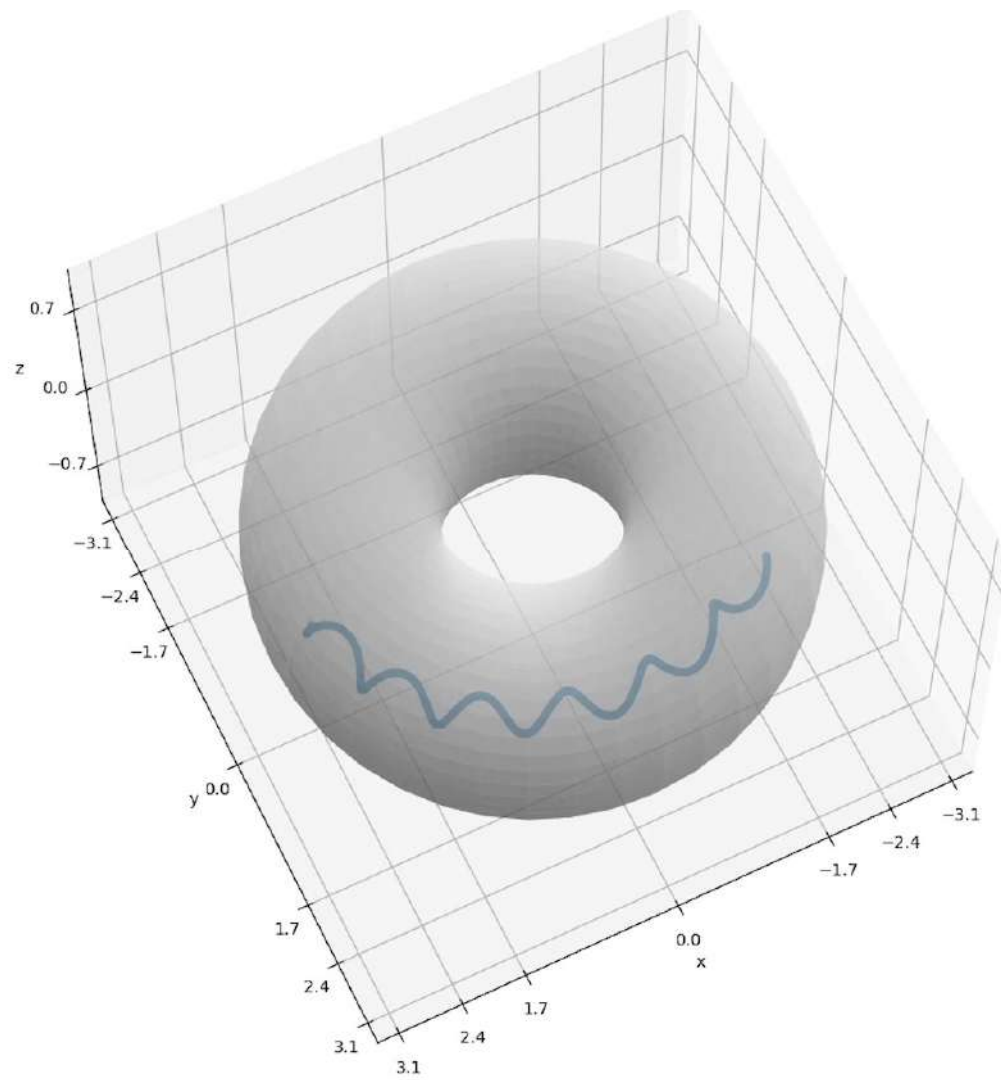


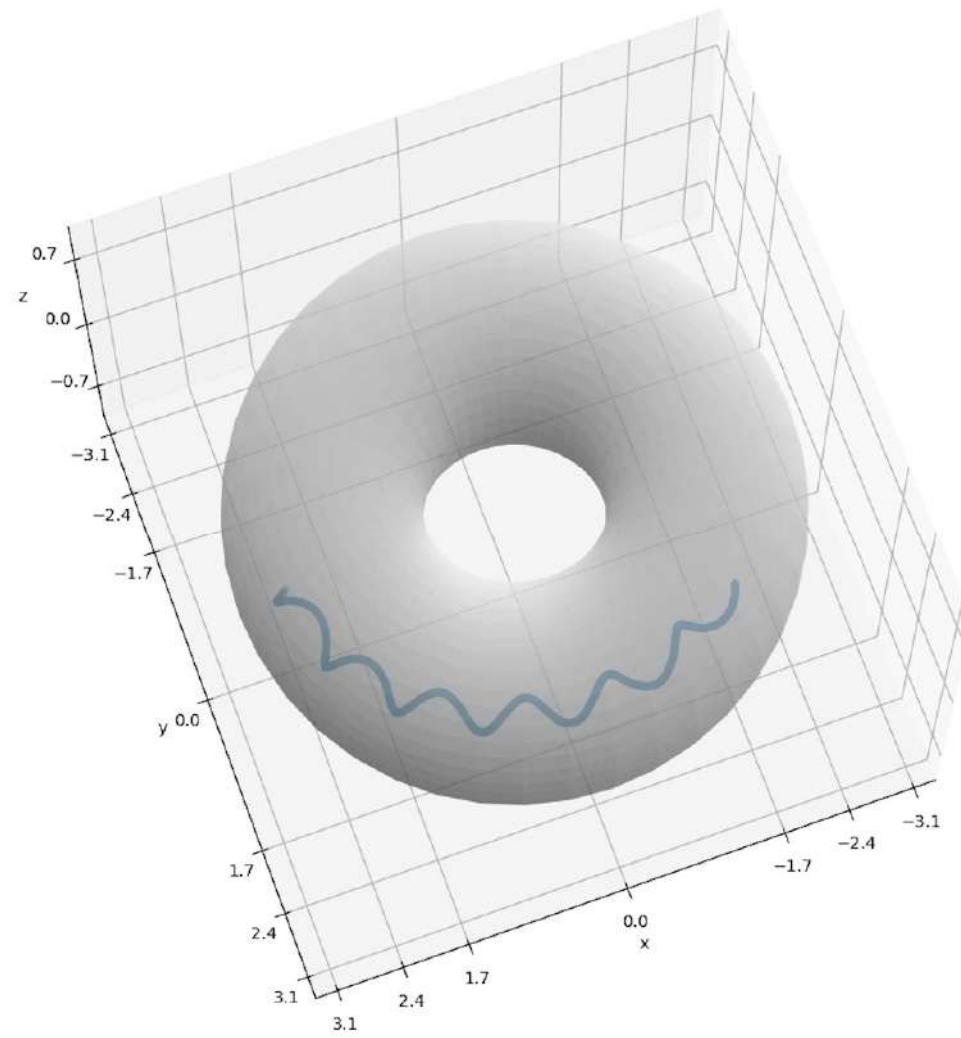


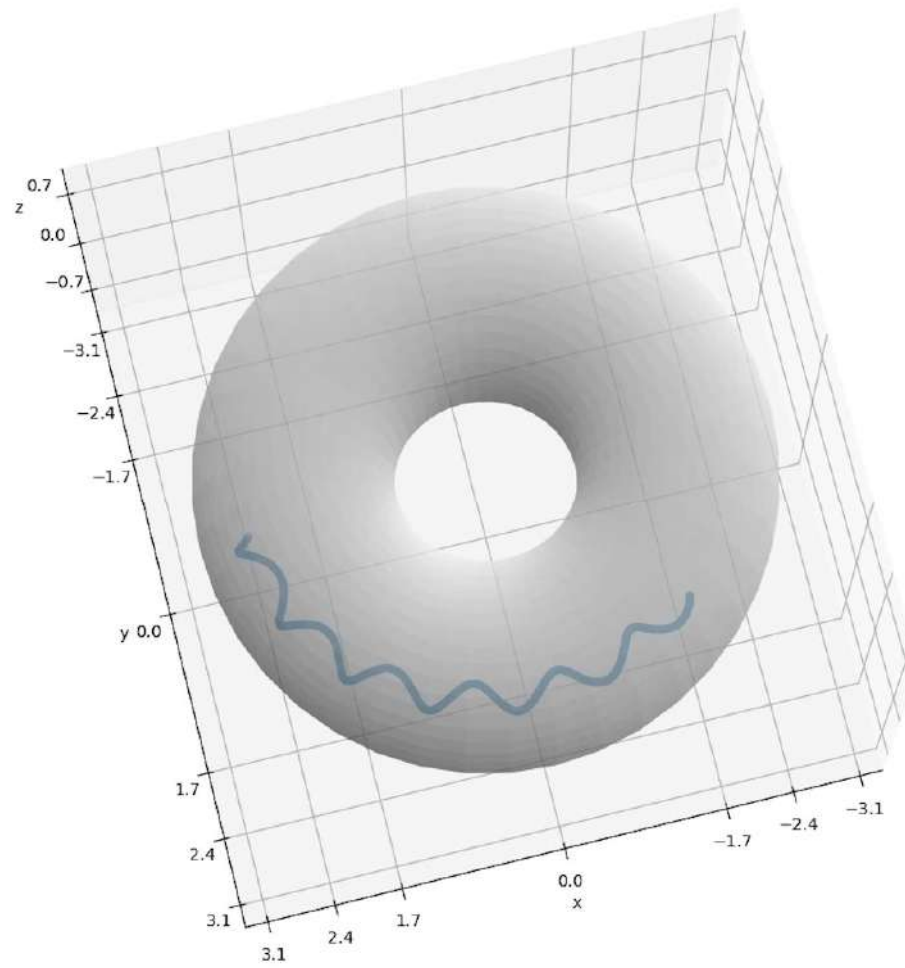


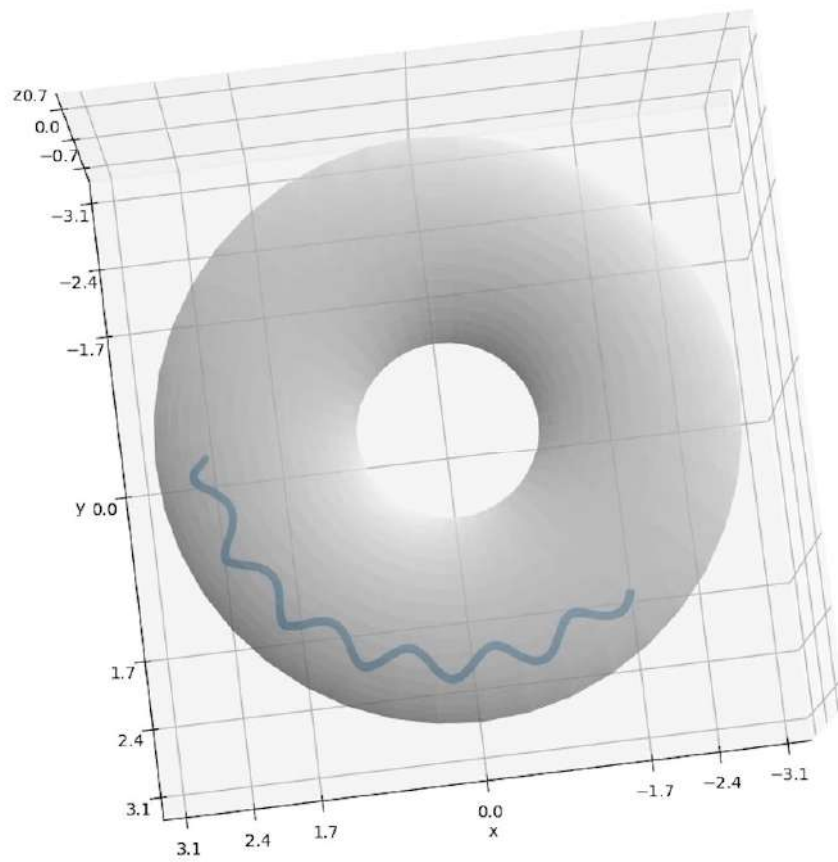


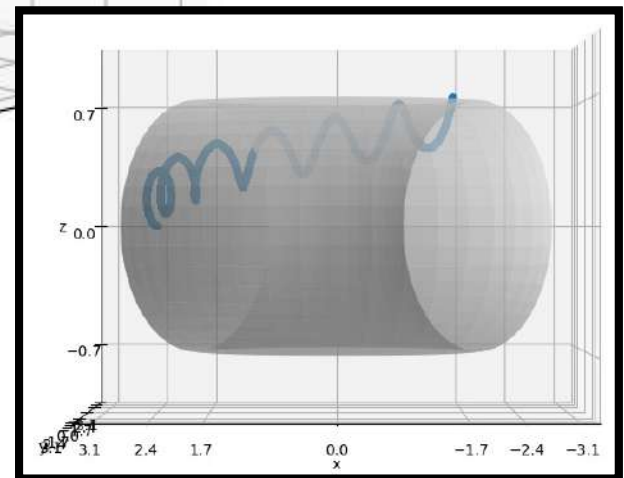
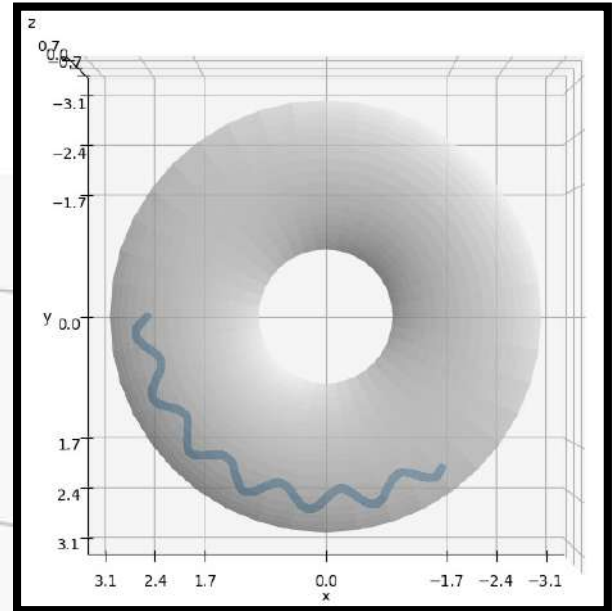
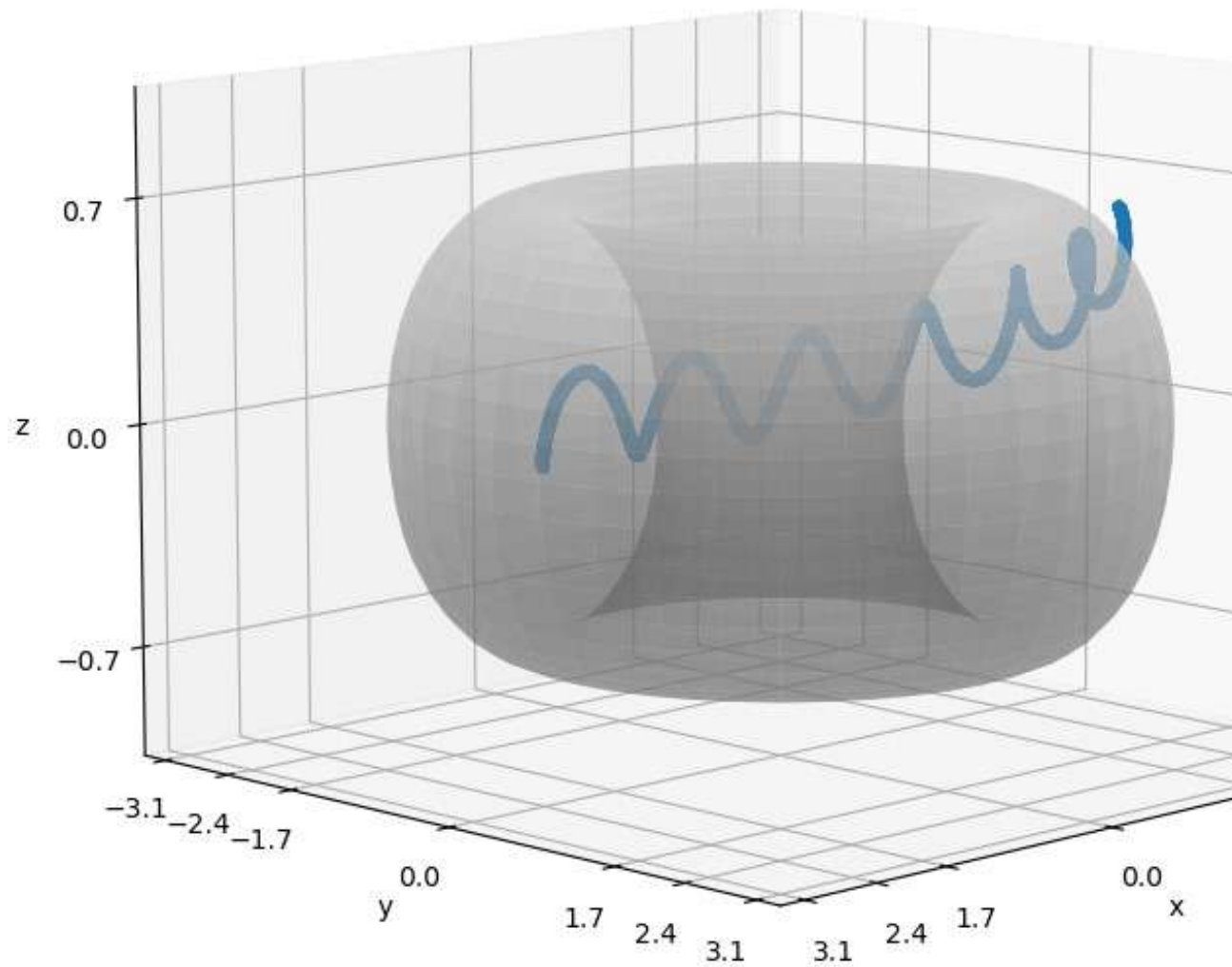






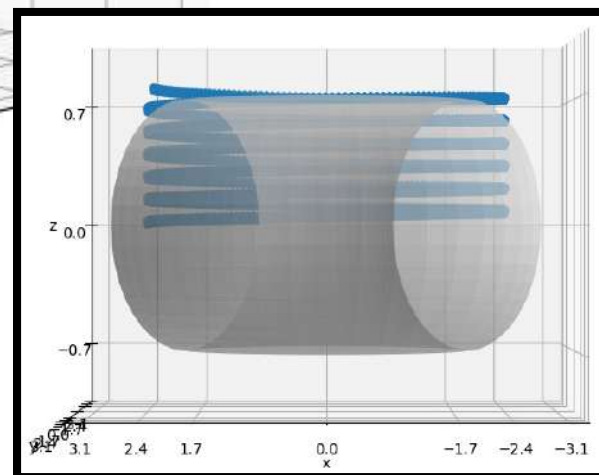
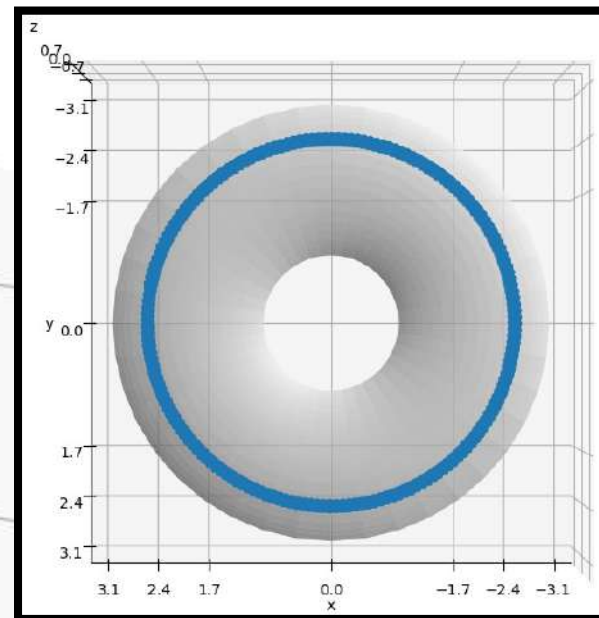
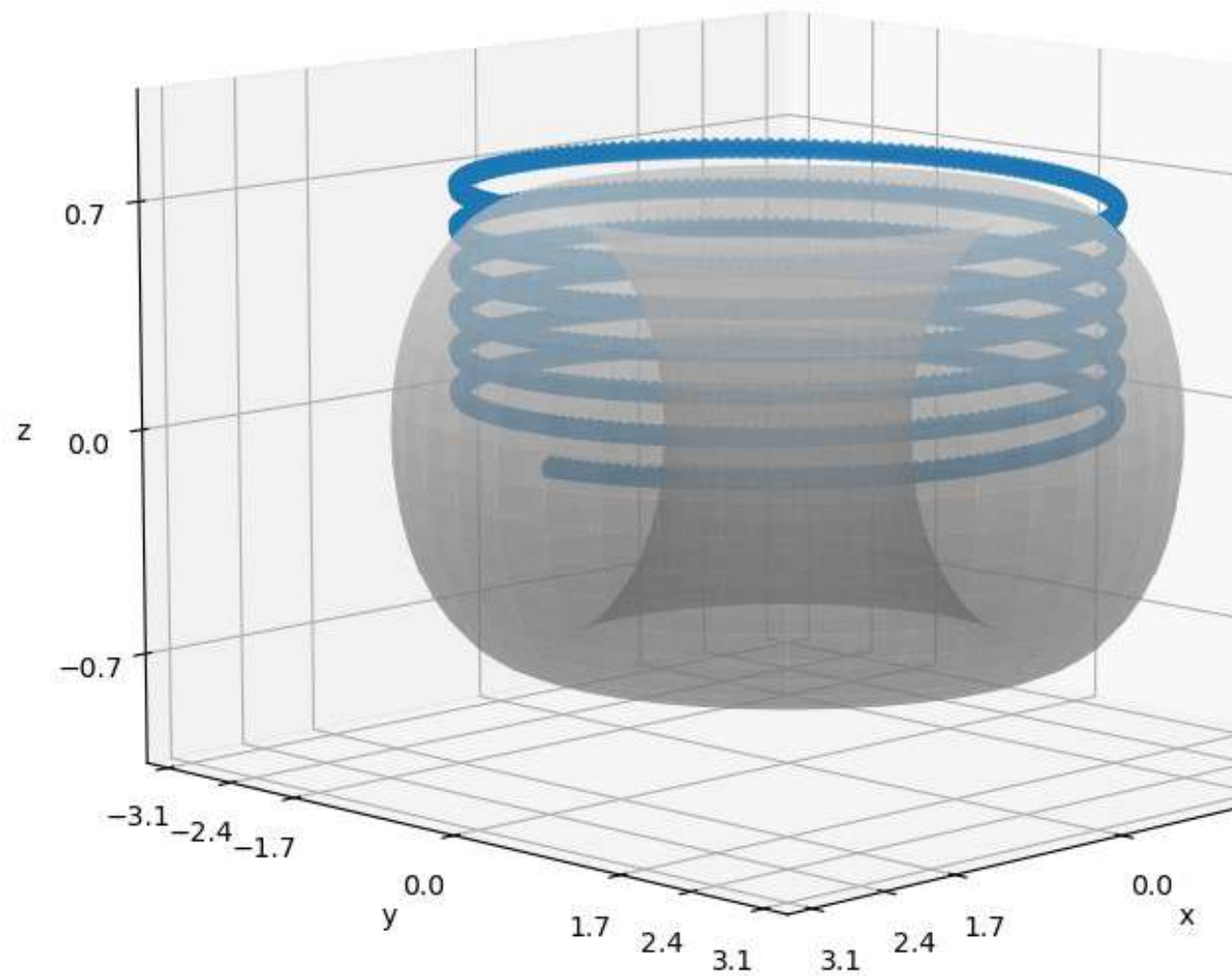




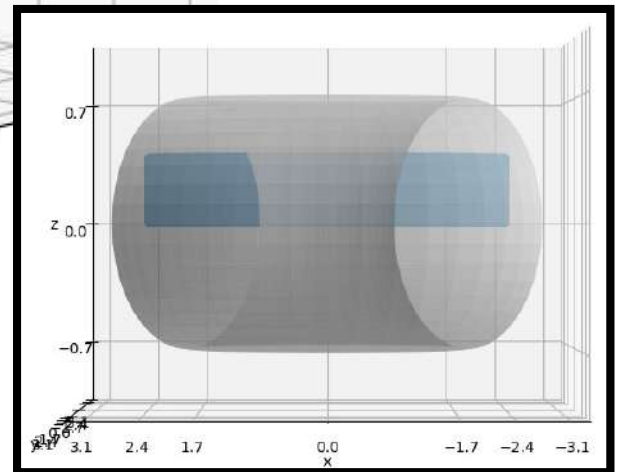
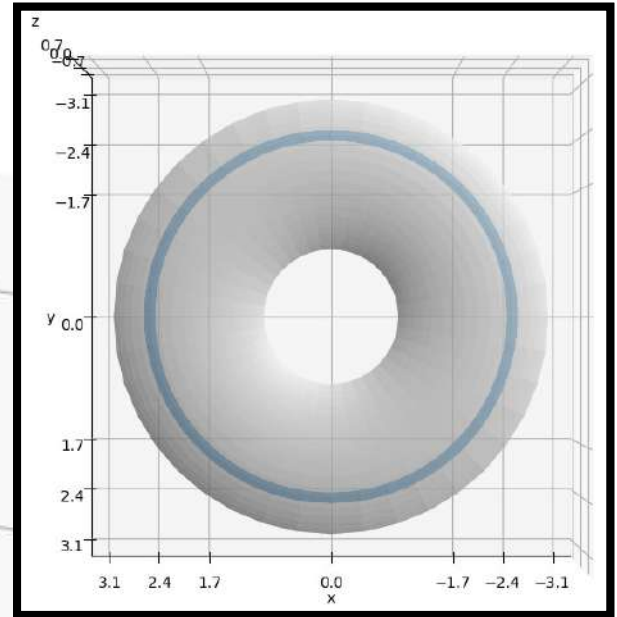
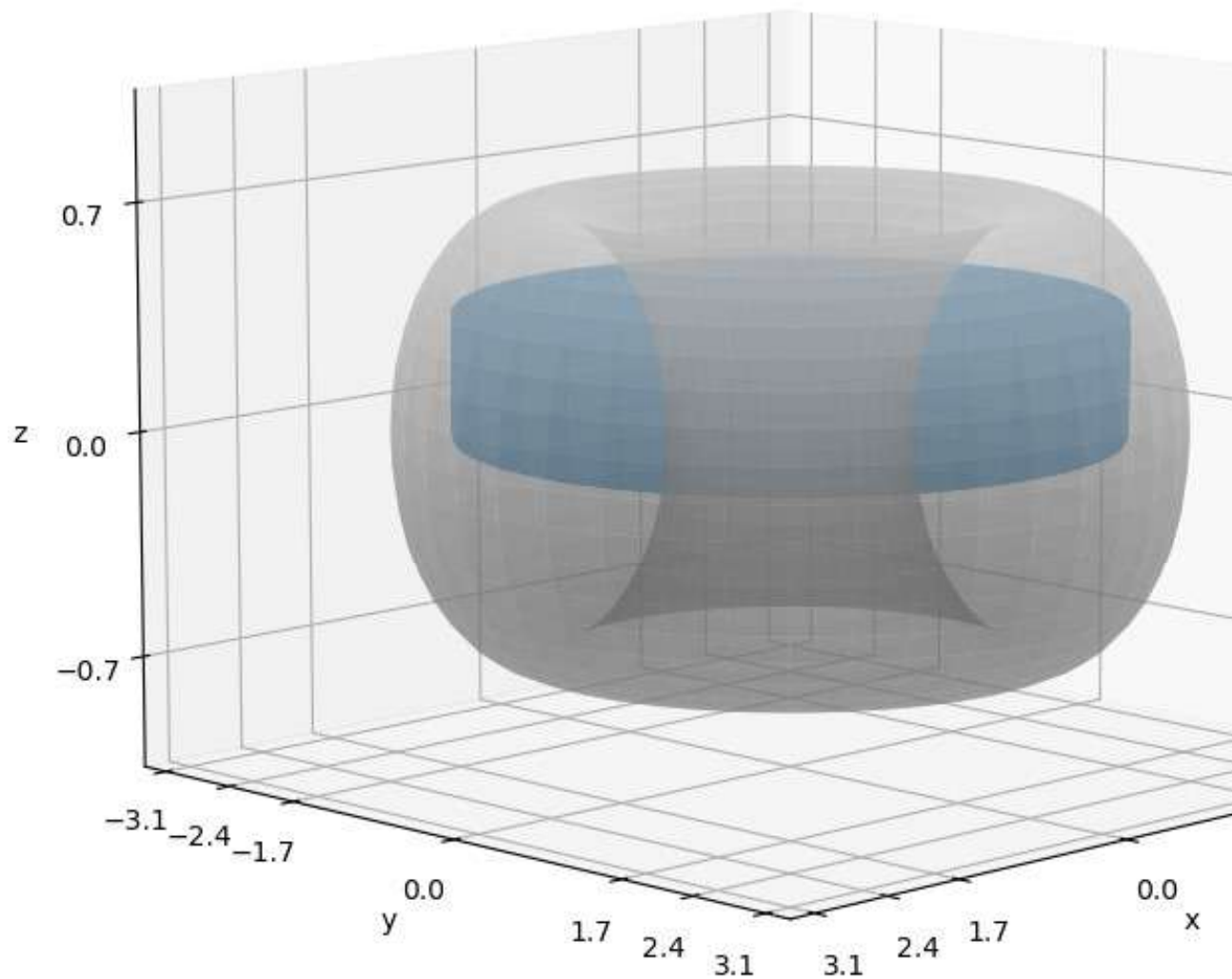


Temps simulé: $5,4 \cdot 10^{-11}$ s

Durée de confinement: $5,4 \cdot 10^{-11}$ s

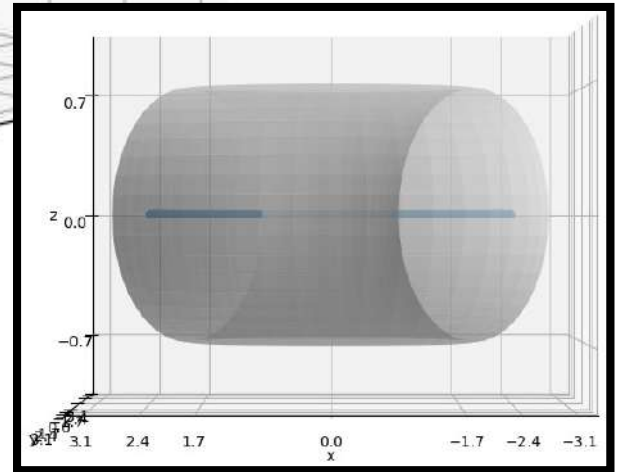
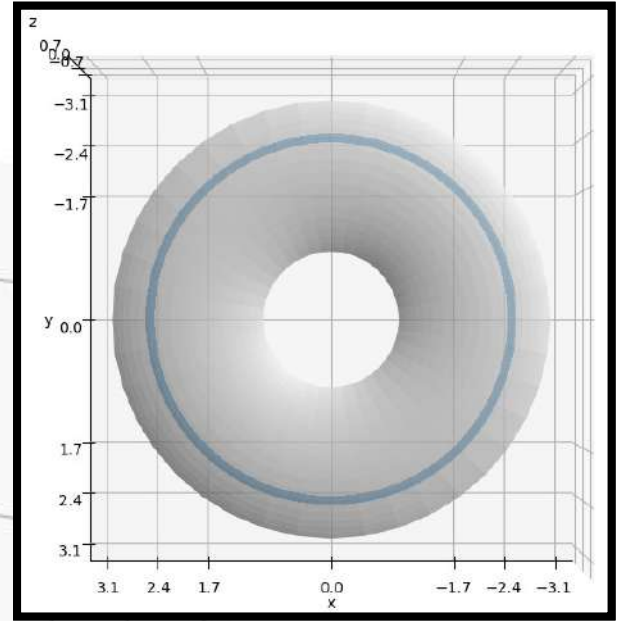
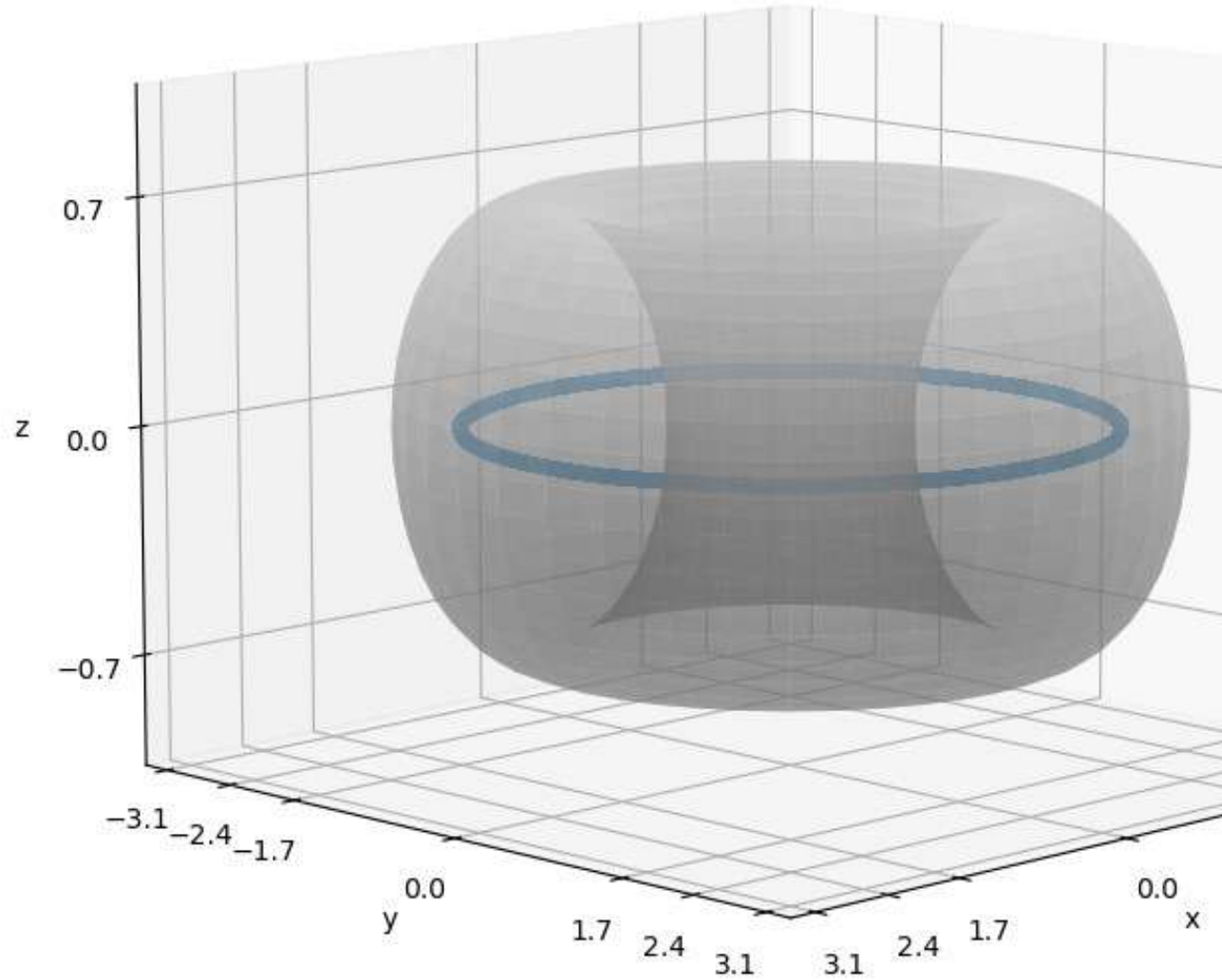


Temps simulé: 10^{-8} s
 Durée de confinement: $8,07 \cdot 10^{-9}$ s



Temps simulé: 10^{-4} s

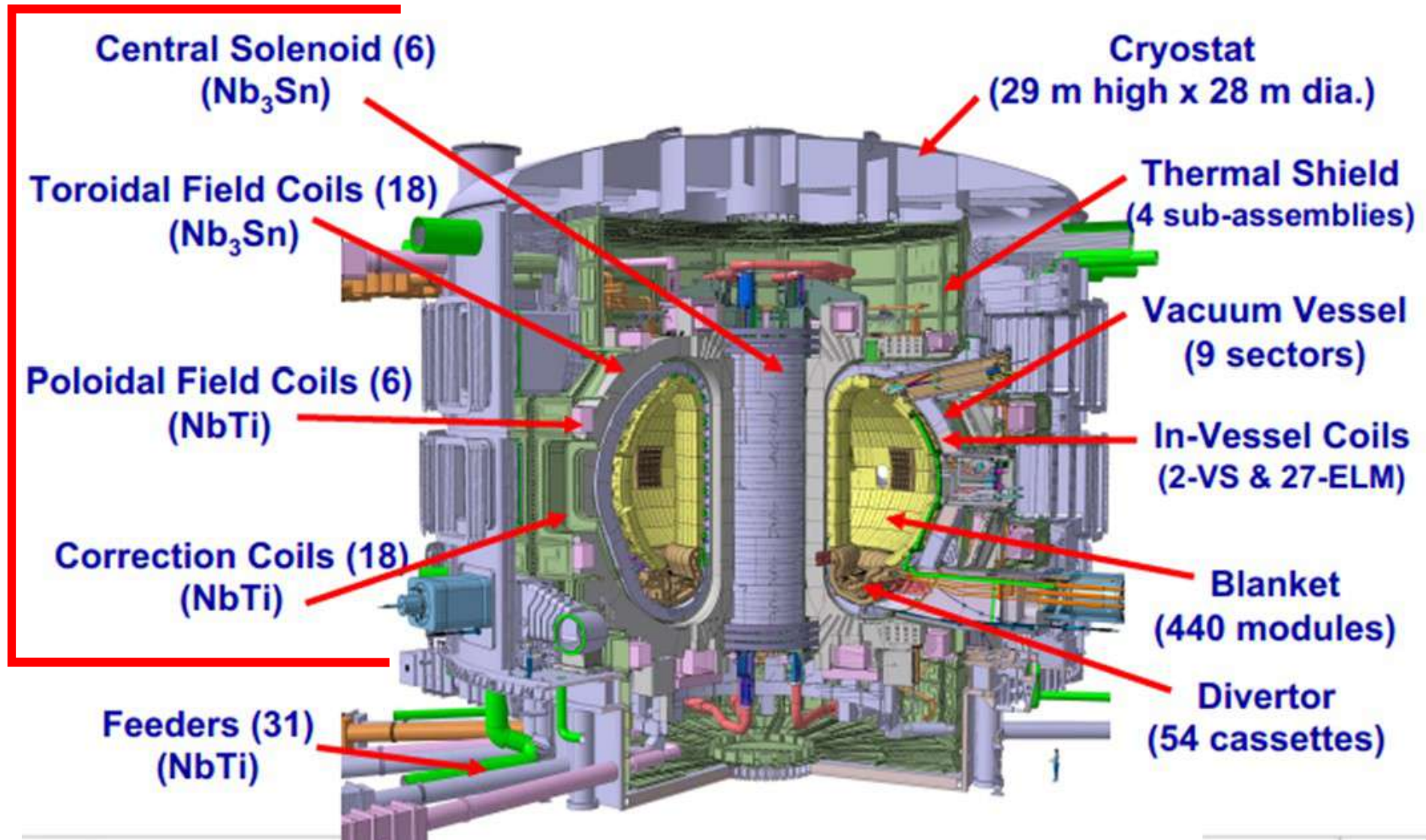
Durée de confinement estimée: $2 \cdot 10^{-4}$ s



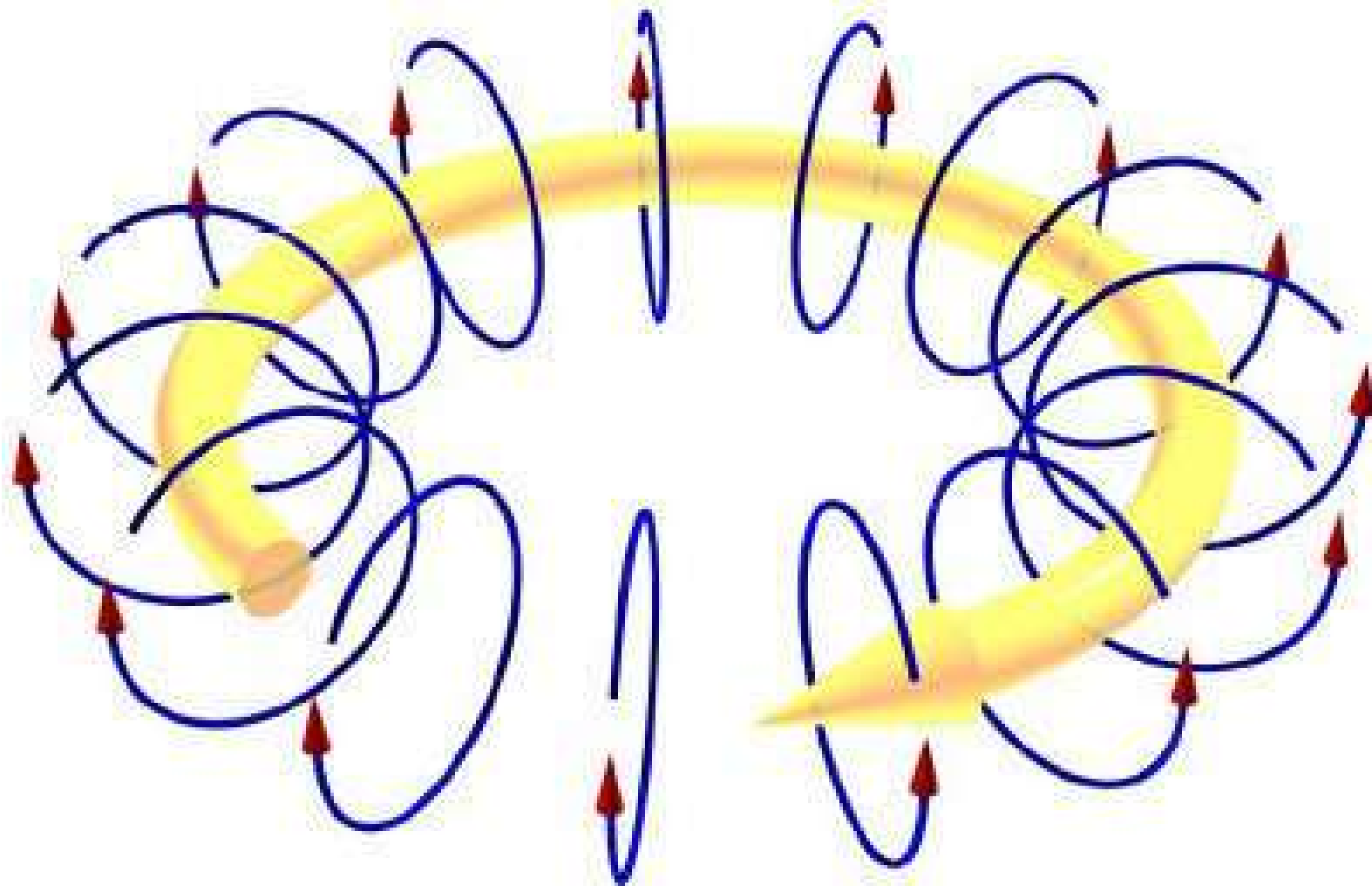
Temps simulé: 10^{-4} s

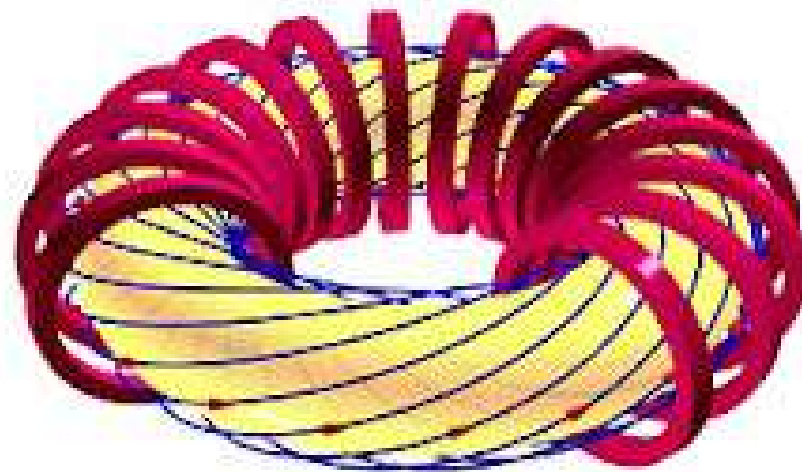
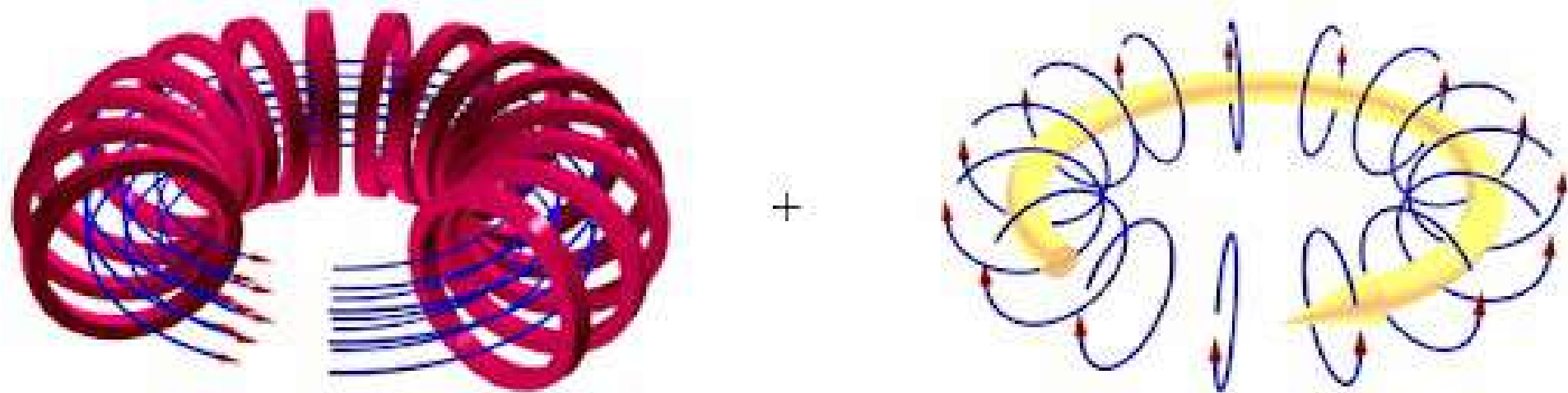
Durée de confinement estimée: 1,17 s

III: Amélioration du Modèle

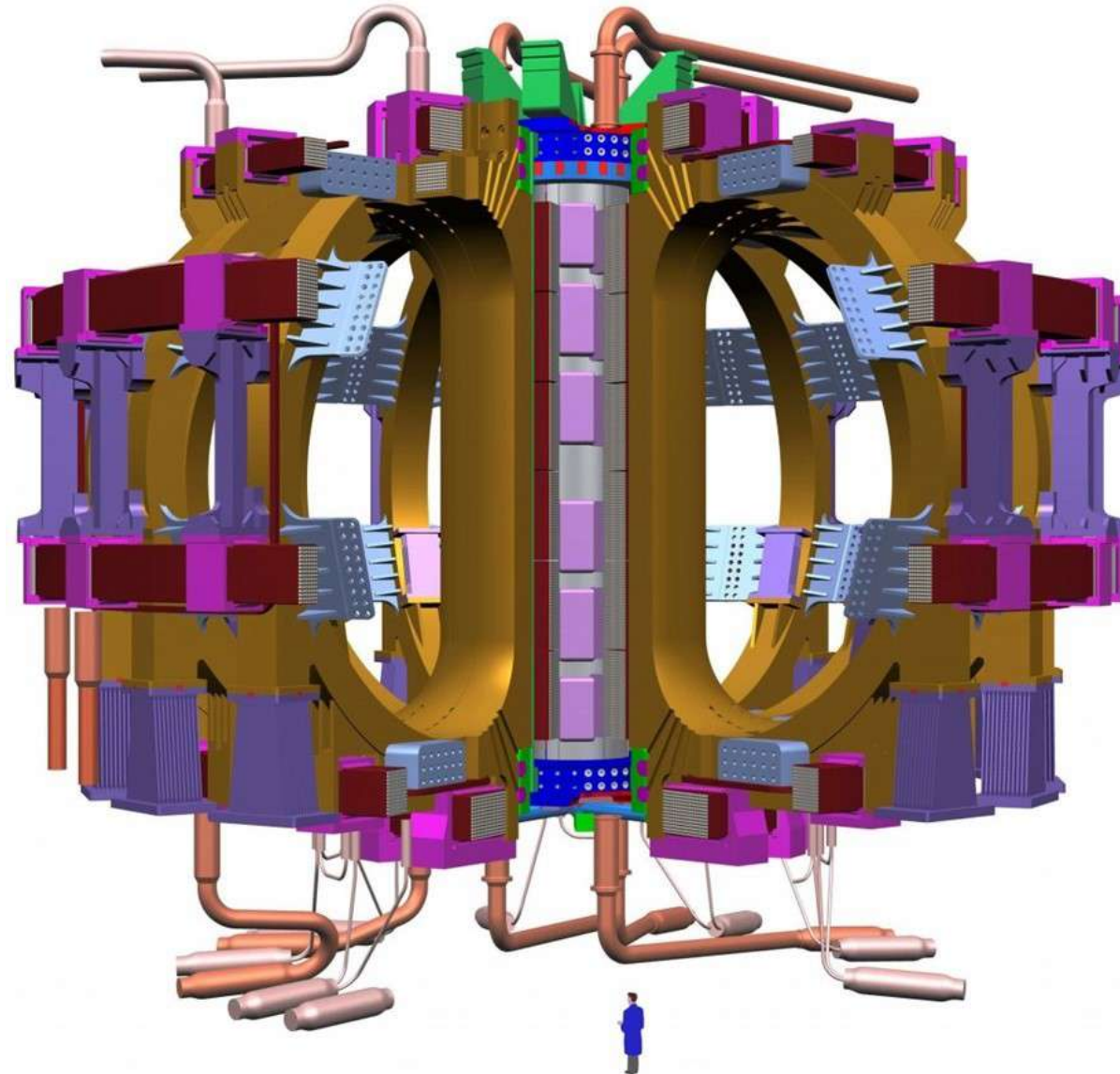


Champ Magnétique Poloidal

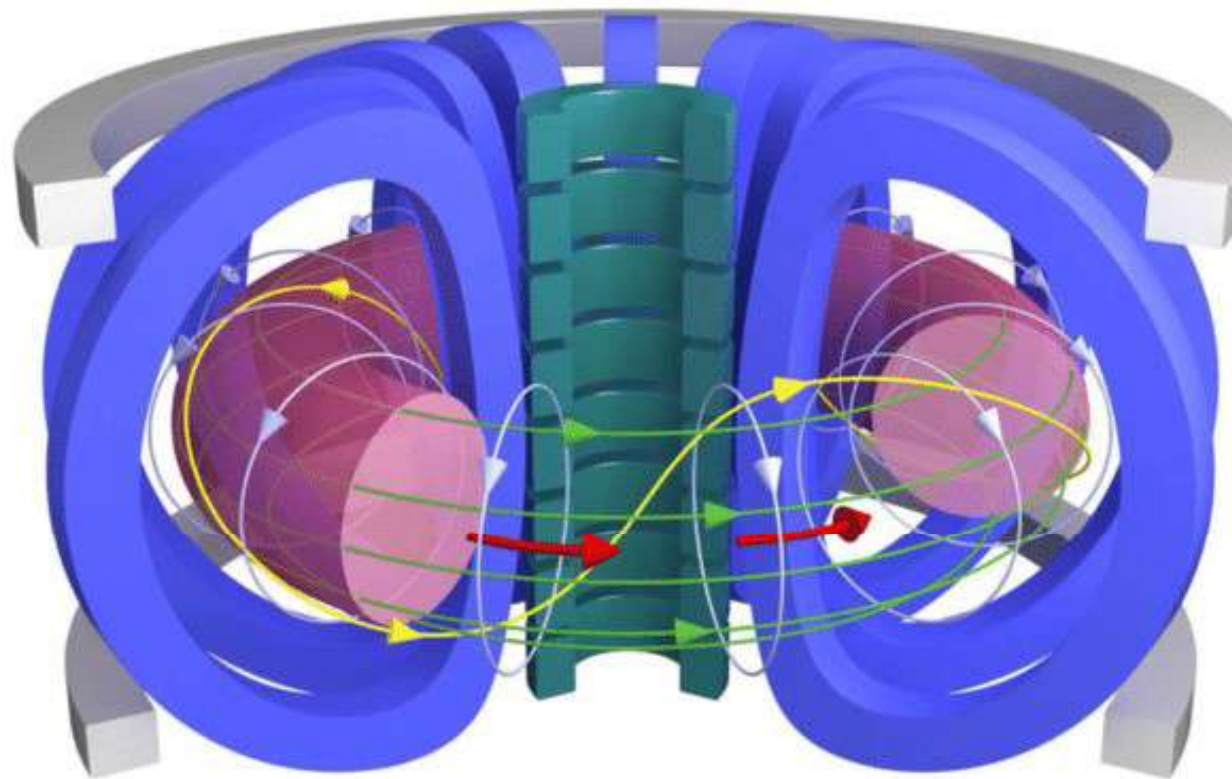












Une première solution: Ajout de bobines horizontales pour créer un champ poloïdal



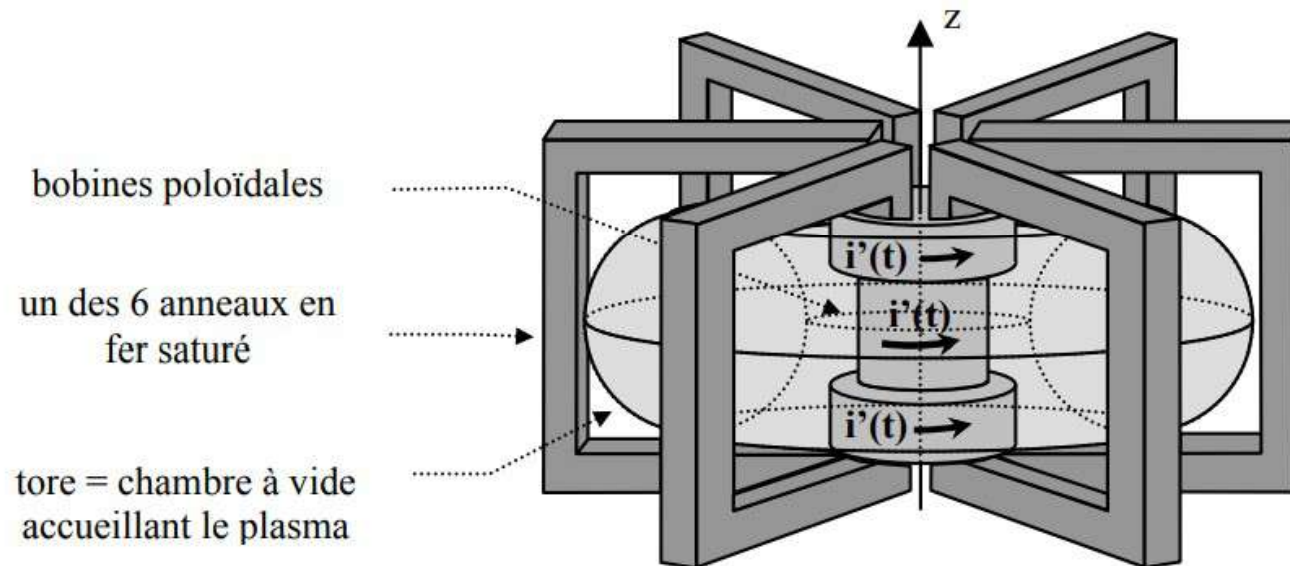
Représentation Schématique du Tokamak ITER



- | | |
|--|--|
|  Bobines de champ toroïdales |  Champ magnétique poloïdal |
|  Plasma confiné |  Champ magnétique toroïdal |
|  Bobines internes de champ poloidal |  Champ magnétique résultant |
|  Bobines externes de champ poloidal |  Courant plasma |

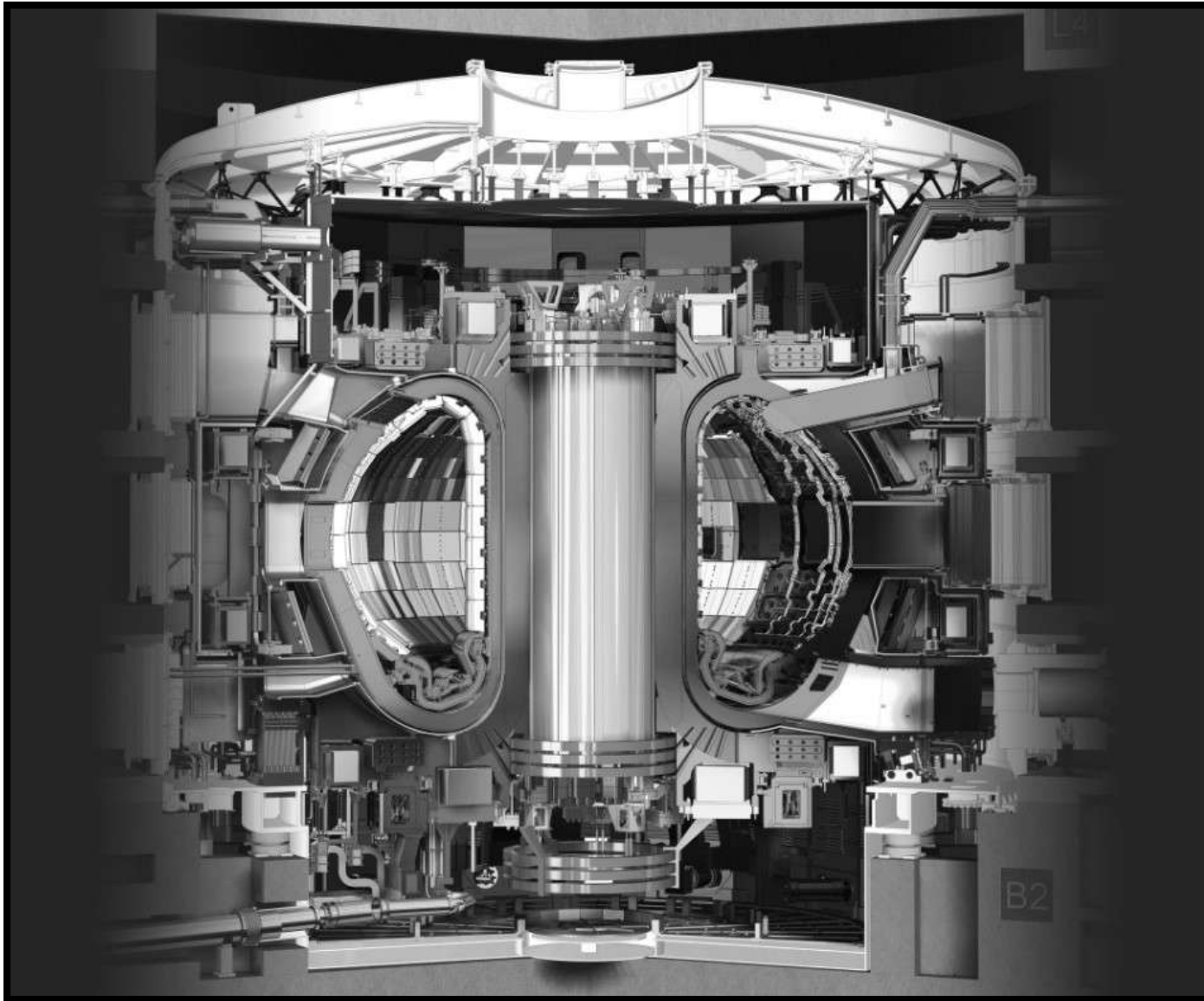
Une seconde solution: Ajout d'anneaux en fer saturé

- 3) Pour contrecarrer cette dérive des particules chargées lorsque seul agit \vec{B}_T , on utilise un jeu de bobines coaxiales au tore, appelées bobines poloïdales, composées d'un grand nombre de spires parcourues par un courant variable $i'(t)$, de fréquence $\nu = 50$ Hz. On associe à ces bobines 6 anneaux rectangulaires en fer saturé, régulièrement répartis autour du tore.

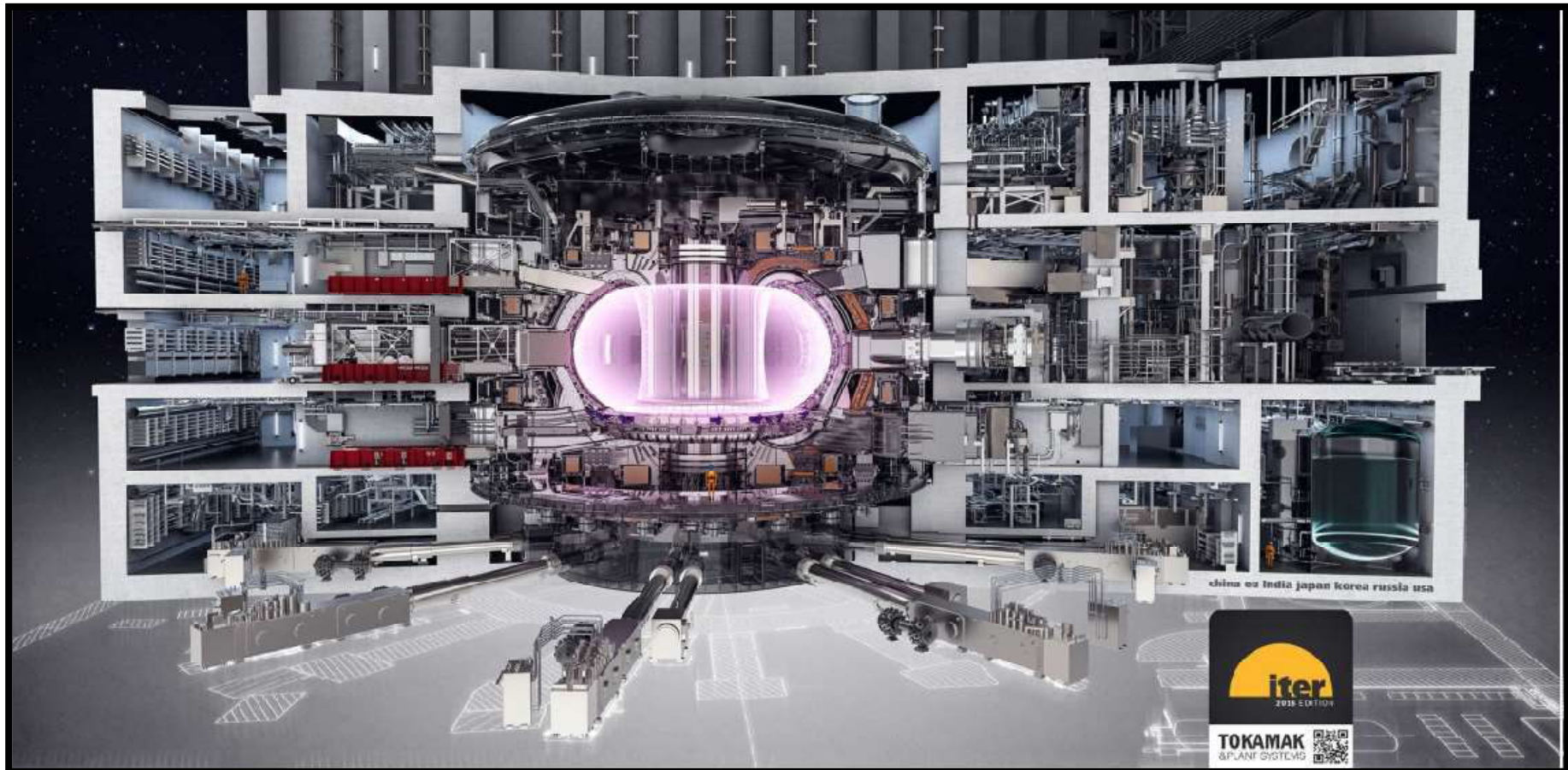


Pour alléger le schéma, les bobines toroïdales n'y sont pas représentées, mais imposent toujours le champ magnétique toroïdal \vec{B}_T aux particules contenues à l'intérieur du tore. Le repérage dans le tore est identique à celui défini sur le schéma de la partie B.

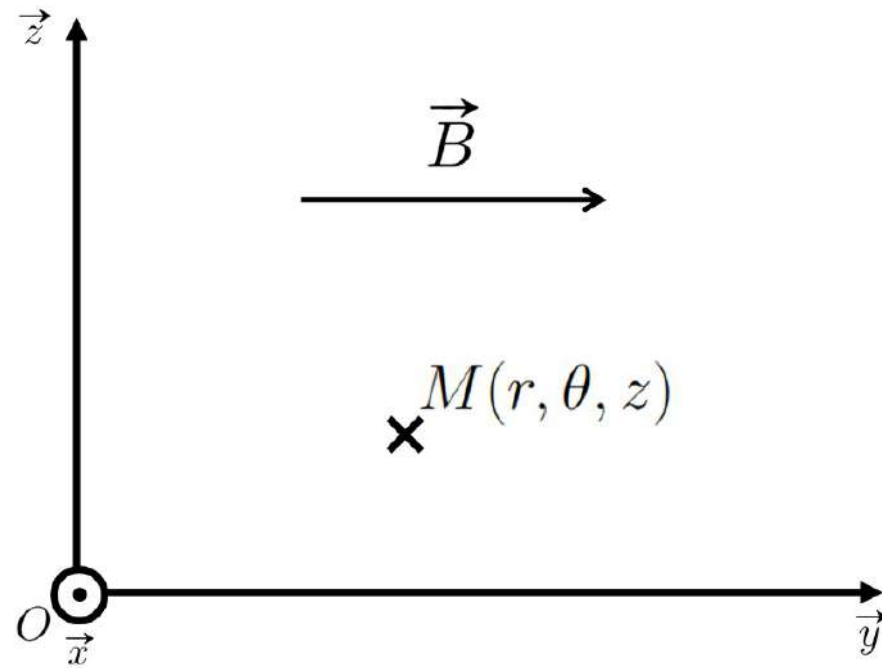
Conclusion



Annexes



Etude d'une particule chargée dans un champ uniforme



Systeme: Particule chargée de charge q et de masse m

Bilan des forces: Force de Lorentz: $\vec{F} = q\vec{v} \wedge \vec{B}$

Principe fondamental de la dynamique: $m\frac{d\vec{v}}{dt} = q\vec{v} \wedge \vec{B}$

$$m \frac{d\vec{v}}{dt} = \epsilon \omega_c \vec{v} \wedge \vec{u}_y \iff \begin{cases} \dot{v}_x = -\epsilon \omega_c v_z \\ \dot{v}_y = 0 \\ \dot{v}_z = \epsilon \omega_c v_x \end{cases}$$

Avec: $\omega_c = \frac{|q|B}{m}$ la pulsation cyclotron, et $\epsilon = \frac{q}{|q|}$

Décomposition de la vitesse : $\vec{v} = \vec{v}_\perp + \vec{v}_//$

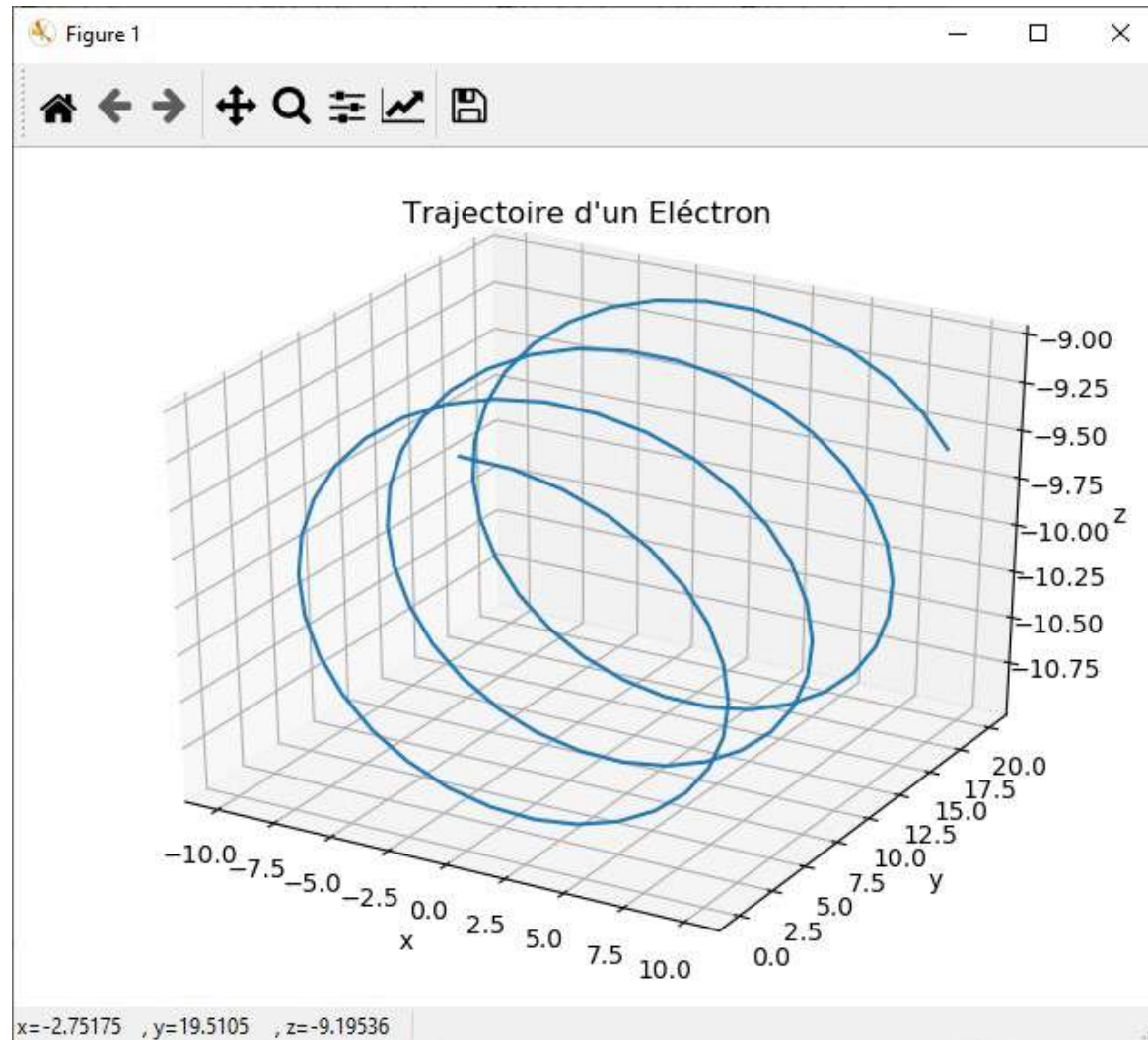
Avec $\vec{v}_//$ et \vec{v}_\perp représentant les composantes parallèle et perpendiculaire au champ \vec{B} .

Conditions initiales: $v_x(0) = v_\perp > 0$, $v_y(0) = v_//$ et $v_z(0) = 0$.

En découplant le système: $v_x + i v_z = v_\perp e^{i\epsilon \omega_c t}$

$$\begin{cases} v_x = v_\perp \cos(\omega_c t) \\ v_y = v_// \\ v_z = \epsilon v_\perp \sin(\omega_c t) \end{cases}$$

Finalement:
$$\begin{cases} x(t) = \rho_L \sin(\omega_c t) + x_0 \\ y(t) = v_{//} t + y_0 \\ z(t) = \epsilon \rho_L (1 - \cos(\omega_c t)) + z_0 \end{cases}$$
 Avec: $\rho_L = \frac{v_{\perp}}{\omega_c}$ le rayon de Larmor

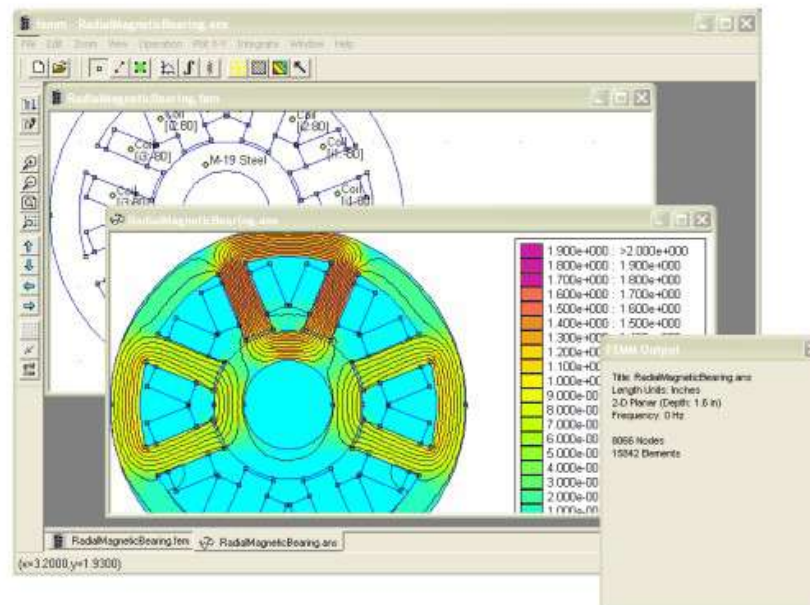


Logiciel FEMM: <https://www.femm.info/wiki/HomePage>

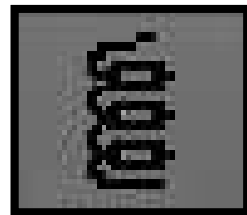
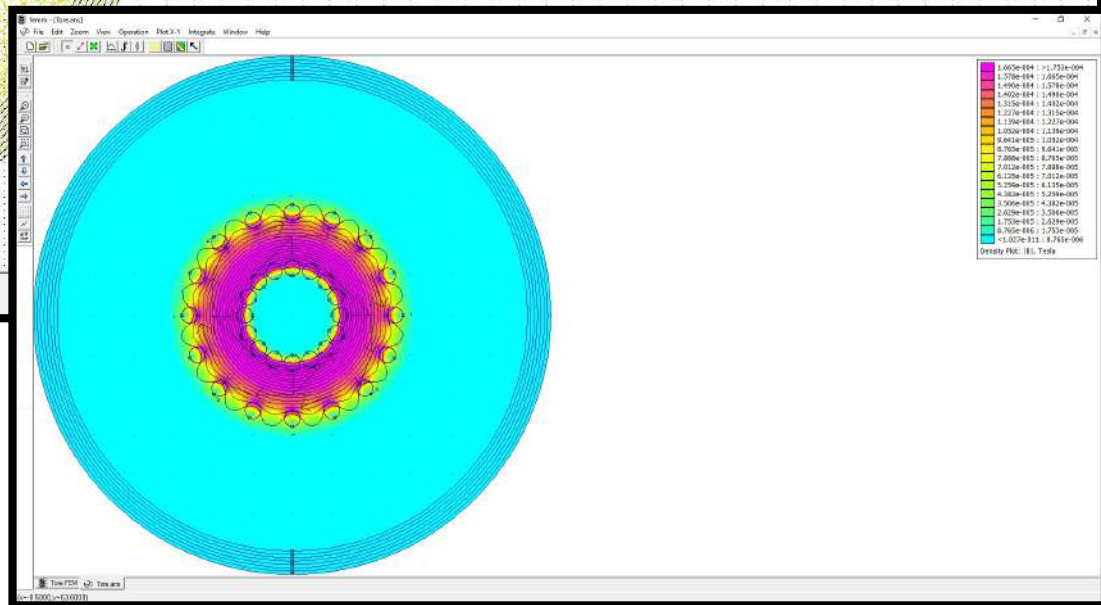
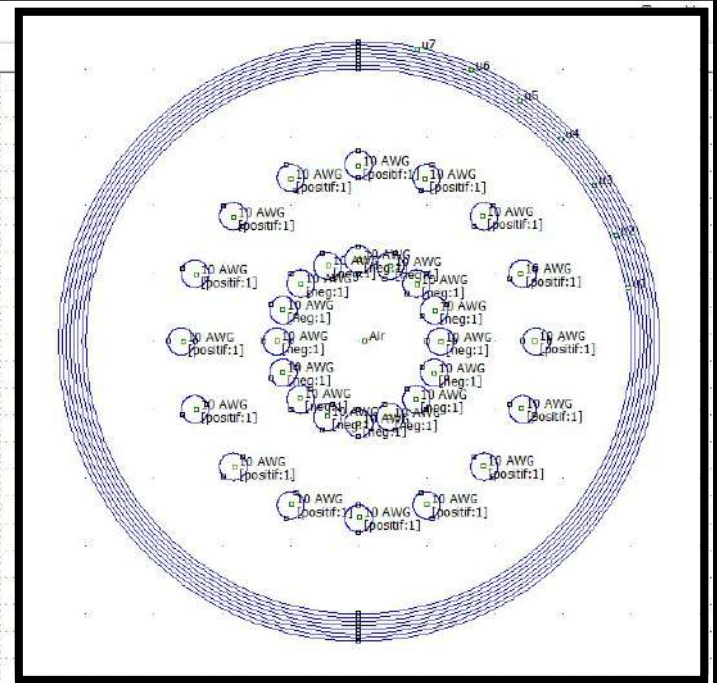
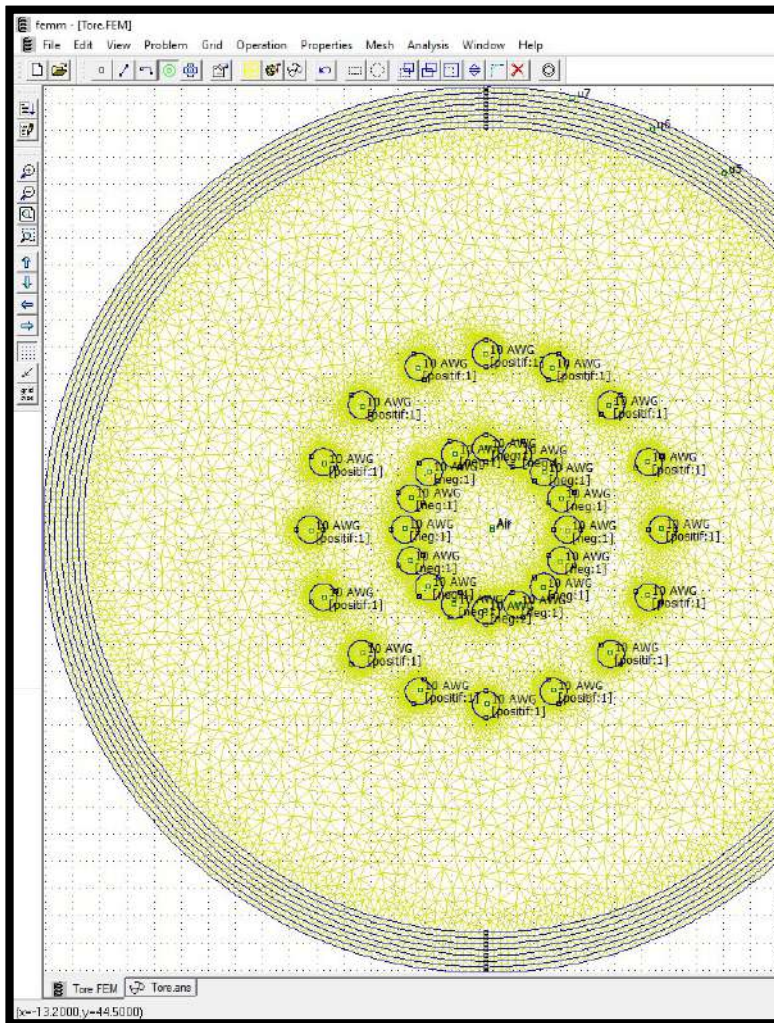
Finite Element Method Magnetics : HomePage

- [Download](#)
- [Documentation](#)
- [FAQ](#)
- [Linux Support](#)
- [Examples](#)
- [User Contributions](#)
- [Miscellaneous](#)
- [Related Links](#)
- [Author](#)

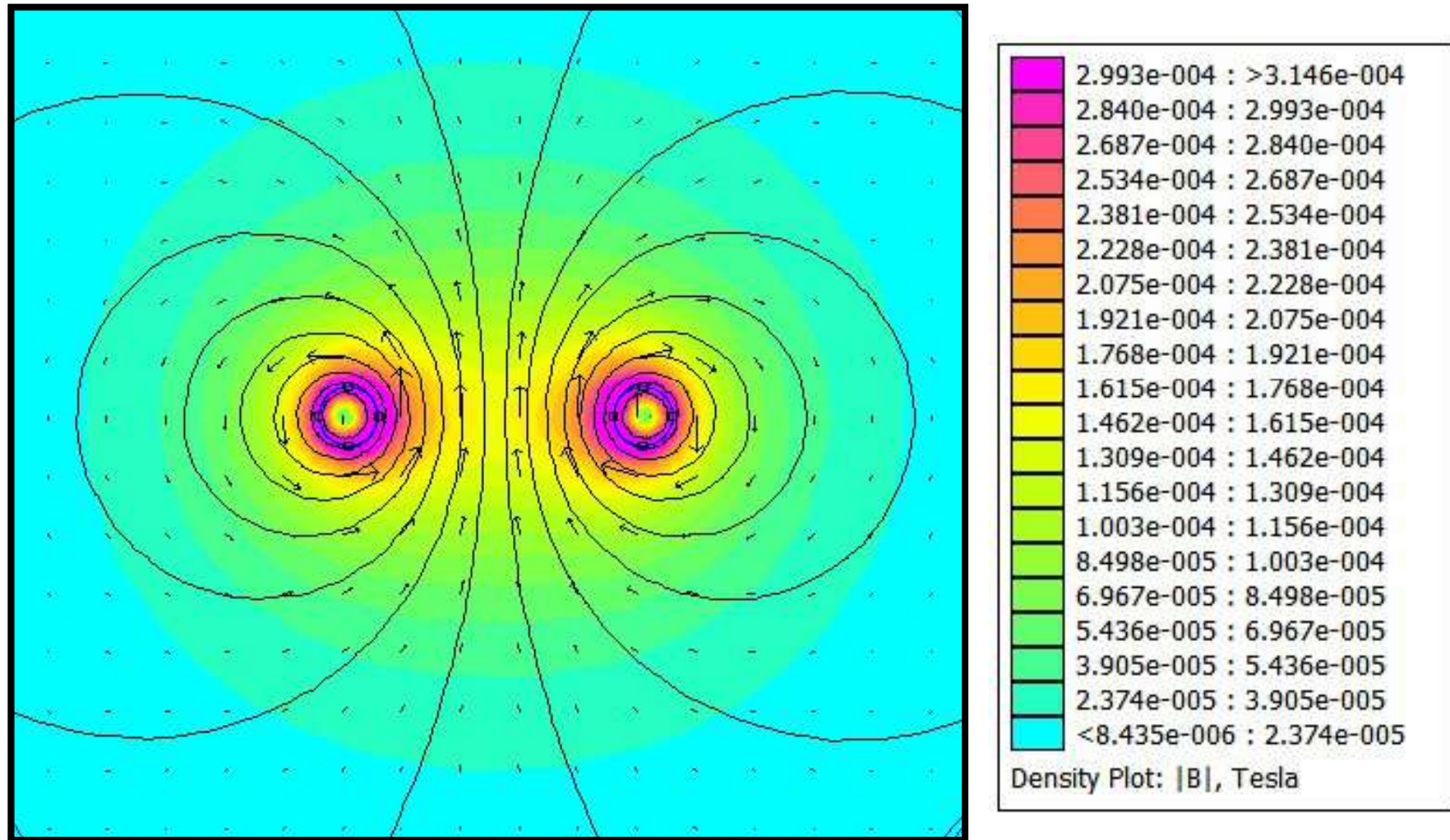
Finite Element Method Magnetics



Valid XHTML :: Valid CSS :: Powered by WikkaWiki



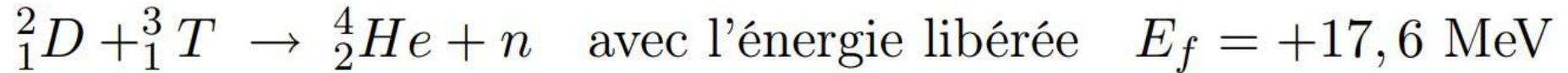
Représentation des lignes de champ d'une spire circulaire avec le logiciel FEMM



(Coupe transversale à la spire)

Choix de la vitesse initiale:

Conservation de la quantité de mouvement



On suppose que toute l'énergie libérée par la réaction de fusion se transforme en énergie cinétique des particules créées, et on néglige les énergies cinétiques des particules incidentes.

Conservation de la quantité de mouvement: $m_n \vec{v}_n + 4m_\alpha \vec{v}_\alpha = \vec{0}$

Avec: $4m_n \approx m_\alpha$, on obtient: $\vec{v}_n = -4\vec{v}_\alpha$

Donc: $E_c^n = 4E_c^\alpha$

Finalement: $E_c^n + E_c^\alpha = 4E_c^\alpha + E_c^\alpha = E_f \Rightarrow E_c^n = \frac{4}{5}E_f$ et $E_c^\alpha = \frac{1}{5}E_f$

Vitesse initiale: $v_0 = 1,3 \cdot 10^4 \text{ m/s}$

Choix de la vitesse initiale:

Thermodynamique statistique

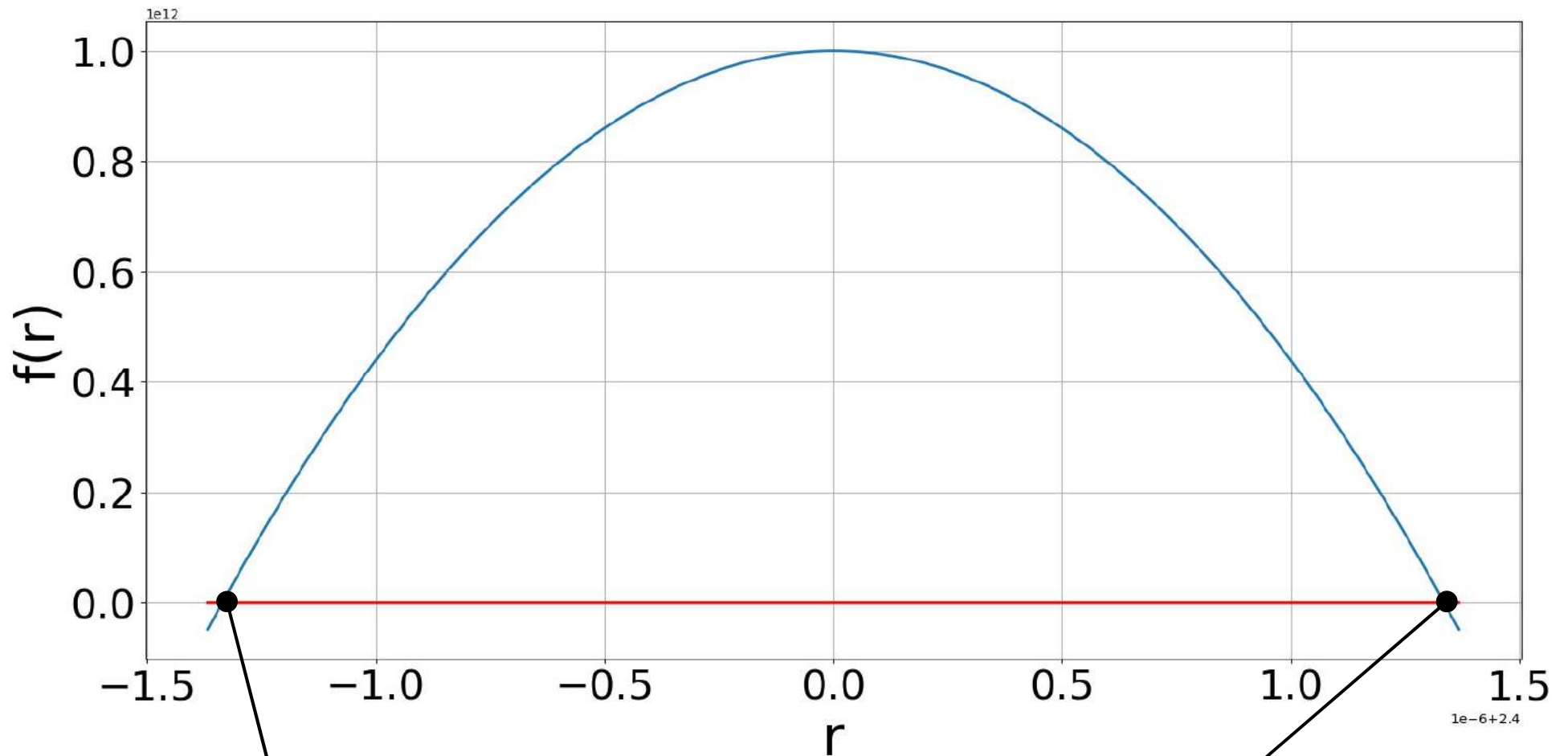
$$\left\langle \frac{1}{2} m v^2 \right\rangle = \frac{3k_B T}{2} \iff \langle v \rangle = \sqrt{\frac{3k_B T}{m}}$$

$$T = 293 \text{ K} \Rightarrow v = 1,2 \cdot 10^5 \text{ m/s}$$

$$T = 10^6 \text{ K} \Rightarrow v = 6,5 \cdot 10^6 \text{ m/s}$$

$$T = 150 \cdot 10^6 \text{ K} \Rightarrow v = 8 \cdot 10^7 \text{ m/s}$$

$$\dot{r}^2 = 2v_0^2 - \frac{(Rv_0)^2}{r^2} - \beta^2 \ln \frac{r^2}{R} = f(r)$$



$$R_1 = 2.3999999$$

$$R_2 = 2.4000001$$

Mise en place d'une procédure numérique pour résoudre par approximation des équations différentielles du premier ordre avec une condition initiale

On pose: $s = \dot{r}$ $Y = \begin{pmatrix} r \\ s \\ \theta \\ z \end{pmatrix}$ $Y_0 = \begin{pmatrix} R \\ v_0 \\ 0 \\ 0 \end{pmatrix}$

$$\dot{Y} = \begin{pmatrix} \dot{r} \\ \dot{s} \\ \dot{\theta} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} s \\ \frac{(Rv_0)^2}{r^3} - \frac{\beta^2}{r} \ln \frac{r}{R} \\ \frac{Rv_0}{r^2} \\ \beta \ln \frac{r}{R} \end{pmatrix} = g(Y)$$

Discretisation du temps: $Y_{n+1} = Y_n + \tau g(Y_n)$

$$\frac{Y_{n+1} - Y_n}{\tau} \approx \dot{Y}(t_n)$$

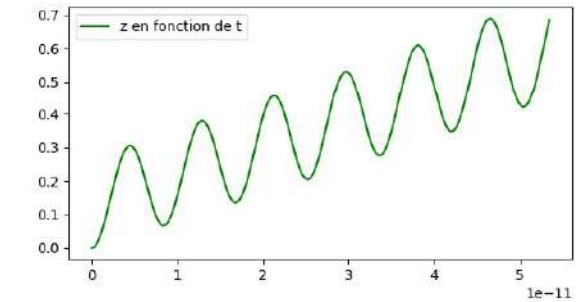
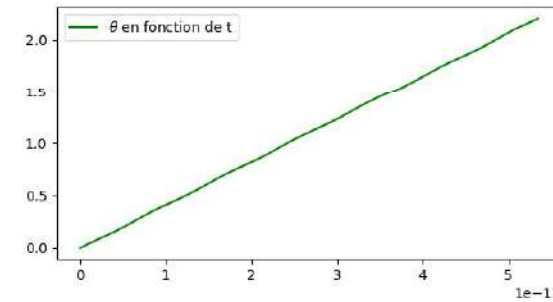
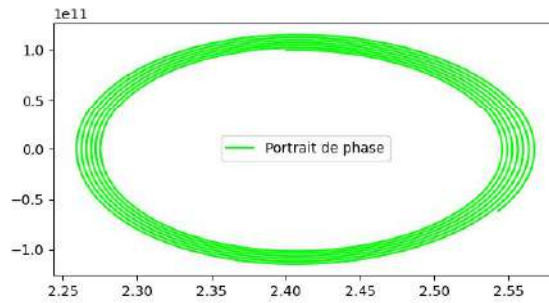
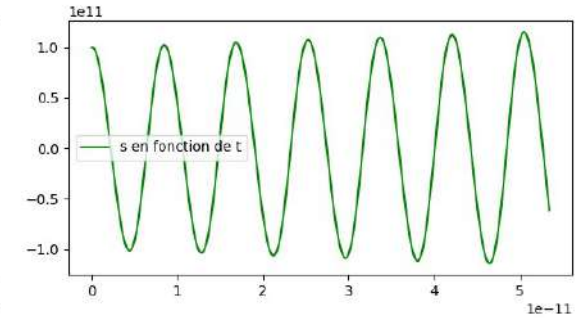
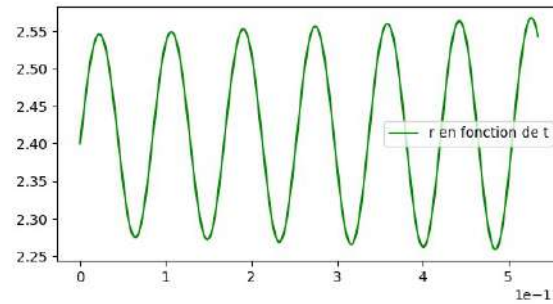
$$\begin{cases} r_0 = R \\ s_0 = v_0 \\ \theta_0 = 0 \\ z_0 = 0 \end{cases}$$

$$\begin{cases} r_{n+1} = r_n + \tau s_n \\ s_{n+1} = s_n + \tau \left(\frac{(Rv_0)^2}{r^3} - \frac{\beta^2}{r} \ln \frac{r}{R} \right) \\ \theta_{n+1} = \theta_n + \tau \frac{Rv_0}{r^2} \\ z_{n+1} = z_n + \tau \beta \ln \frac{r}{R} \end{cases}$$

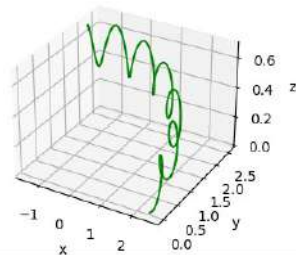
Résultats Obtenus avec la Méthode d'Euler

Méthode d'Euler

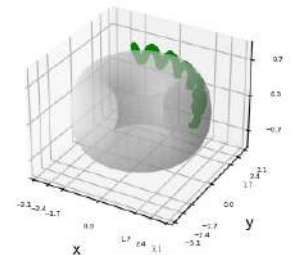
	Valeur Numérique	Unité
Grand rayon R	2.4	m
Petit rayon p	0.7	m
Intensité courant bobines I	1400	A
Nombre de Bobines	35504	
Charge particule q	1.6000000000000000e-19	C
Masse particule m	9.1e-31	kg
Constante β	1797120000000.0	
Vitesse initiale particule	10000000000.0	m/s
Coordonnées initiales de la particule	(2.4, 0, 0)	(m, rad, m)
Intervalle de temps maximal	100	
Pas utilisé dans la méthode d'Euler	1e-14	
Collision au bout de	4350	étapes
Collision au bout de	4.35e-11	s



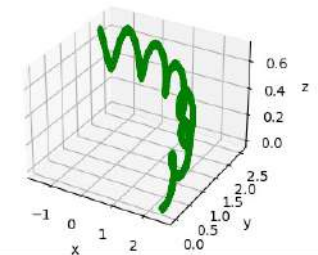
Trajectoire d'un Électron



Tore



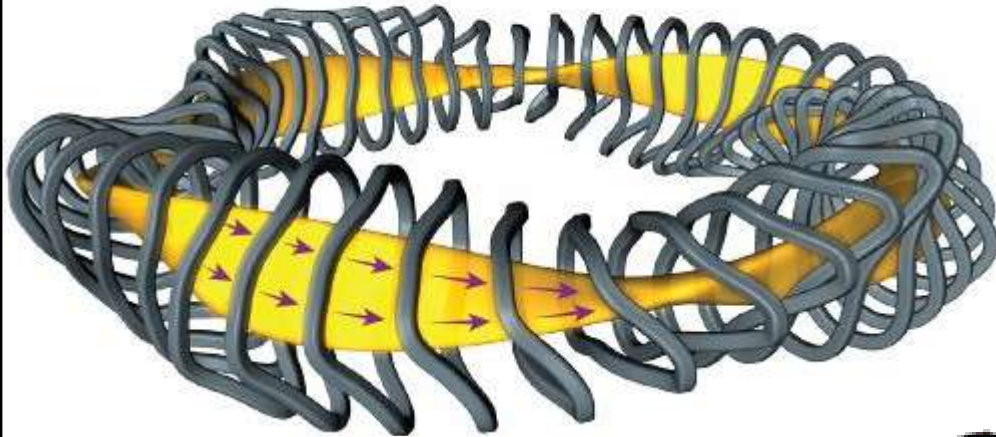
Positions de la particule



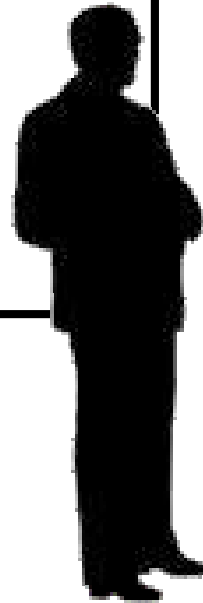
x=5.12e-11 y=2.492

Autres systèmes de confinement magnétique:

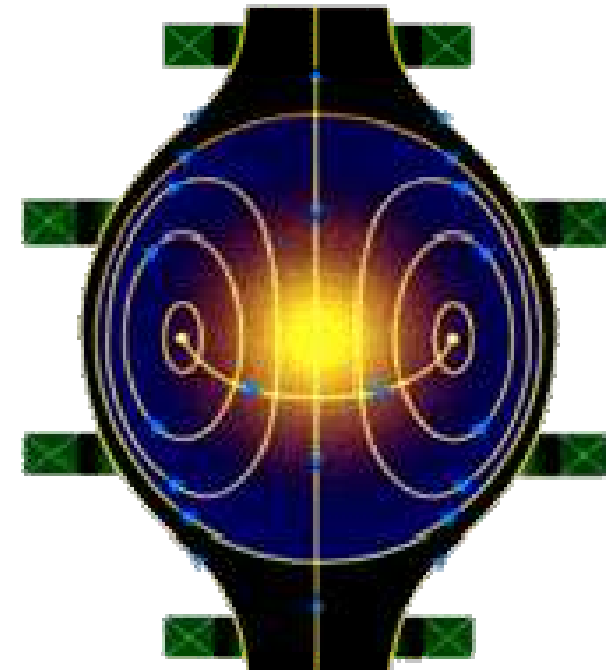
Stellarator



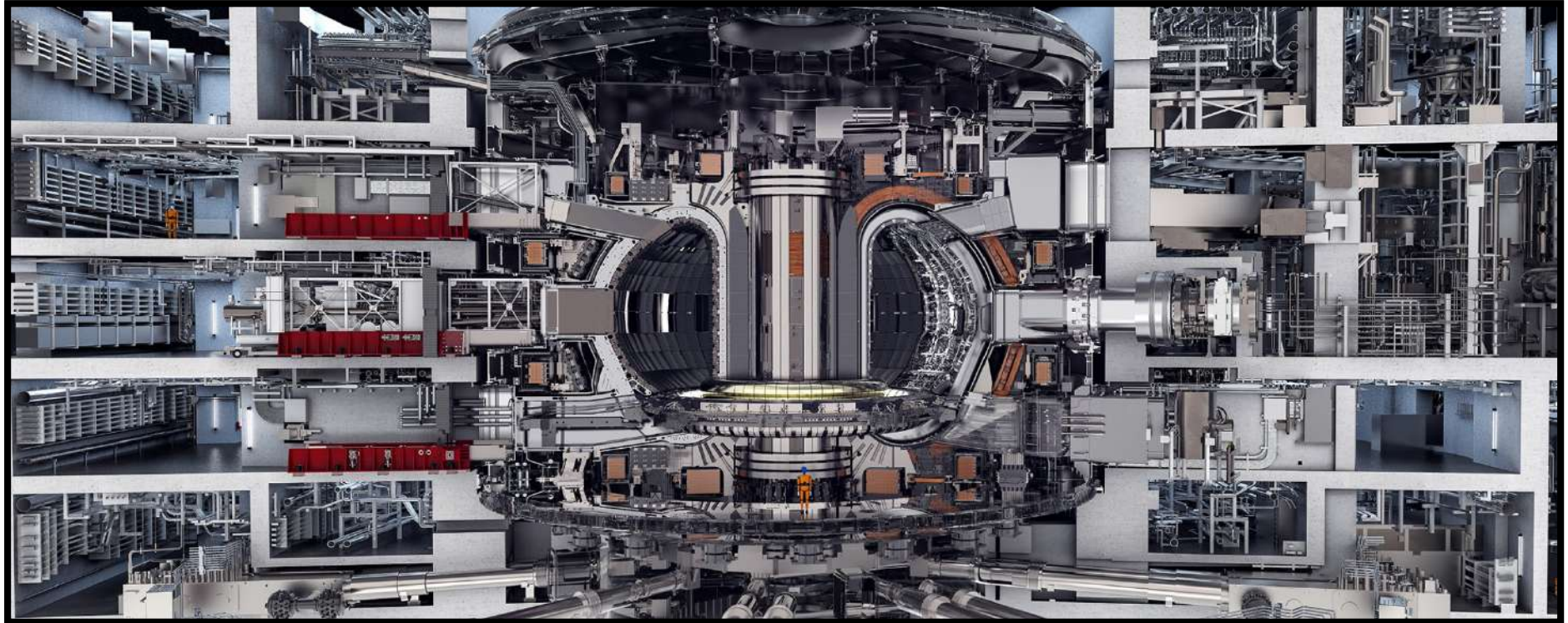
A complex asymmetric magnetic field ensures every plasma particle feels the same force, so the plasma can be maintained - CONTINUOUS OPERATION



Spheromak



Programmes Python



```

001 | ## TIPE:
002 |
003 | ## Modules et Constantes:
004 |
005 | import matplotlib.pyplot as plt
006 | from math import *
007 | import numpy as np
008 | from mpl_toolkits.mplot3d import axes3d
009 | import copy as c
010 |
011 | class color:
012 |     PURPLE = '\033[95m'
013 |     CYAN = '\033[96m'
014 |     DARKCYAN = '\033[36m'
015 |     BLUE = '\033[94m'
016 |     GREEN = '\033[92m'
017 |     YELLOW = '\033[93m'
018 |     RED = '\033[91m'
019 |     BOLD = '\033[1m' # Euler
020 |     UNDERLINE = '\033[4m'
021 |     END = '\033[0m'
022 |
023 | #Paramètres tokamak:
024 | R=2.40 #Grand rayon du tore
025 | rho=0.7 #Rayon du tore
026 | I=1400 #Intensité du courant dans les bobines
027 | N=18*2028 #Nombre de spires
028 | #print(N)
029 |
030 | #Constantes physiques:
031 | q=1.6*(10)**-19 #Charge élémentaire
032 | mu=4*pi*10**-7 #Perméabilité du vide
033 | m=9.1*10**-31 #Masse e-
034 |
035 | B=(q*mu*N*I)/(2*pi*m)
036 | #print(B)
037 |
038 | #Conditions Initiales:
039 | #v0=10**11
040 | #v0=1.3*10**4 #Conservation de la quantité de mouvement
041 | #v0=10**7
042 | #v0=8*10**7 #D'après Thermodynamique (T=150*10**6K)
043 | v0=10**6
044 | t0=0
045 | r0=R
046 | theta0=0
047 | z0=0
048 |
049 | #Paramètres de résolution:
050 | h=10**-14 #Pas utilisé dans les méthodes d'Euler et de Heun
051 | Tmax=10**-10 #Durée maximale de résolution pour les méthodes d'Euler et de Heun
052 |
053 | p_graphe=10**-3
054 | p_dicho=10**-10
055 |
056 | ## Fonction f:
057 |
058 | def f(r):
059 |     #fonction f issue du syst.
060 |     return 2*(v0**2)-((R*v0)**2/(r**2))-(B*log(r/R))**2
061 |
062 | def fonction_test(x):
063 |     #une fonction affine pour effectuer des tests
064 |     return(2*x-1)
065 |
066 | def dichotomie(f,r1,r2,p):
067 |     #Prend en argument une fonction f, un intervalle [r1,r2] et une précision et détermine le point où f
068 |     #s'annule avec la précision p
069 |     while r2-r1>p:
070 |         m=(r1+r2)*0.5
071 |         if f(m)*f(r1)<0:
072 |             r2=m
073 |         else:
074 |             r1=m
075 |     return((r1+r2)*0.5)
076 |
077 | def valeurs_negatives(f,R):
078 |     #Prend en argument une fonction f et R et détermine deux valeurs à gauche et à droite de R pour
079 |     #lesquelles la fonction f est négative
080 |     x1,x2=R,R
081 |     while f(x1)>0:
082 |         x1=x1/2
083 |     while f(x2)>0:
084 |         x2=2*x2
085 |     return(x1,x2)

```

```

086 |
087 | def zone_tore(f,R,p):
088 |     #Prend en argument une fonction f, le grand rayon du tore R et une "précision" p et détermine
l'intervalle de r que peut atteindre la particule
089 |     x1,x2=valeurs_negatives(f,R)
090 |     x1=dichotomie(f,x1,R,p)
091 |     x2=dichotomie(f,R,x2,p)
092 |     return(x1,x2)
093 |
094 | def graphe_f(x1,x2,p):
095 |     #Trace la courbe de f entre x1 et x2 avec un pas p
096 |     X=np.arange(x1,x2,p)
097 |     Y=[]
098 |     Z=[0 for i in range(len(X))]
099 |     for i in X:
100 |         Y.append(f(i))
101 |     plt.figure("Graphe de f")
102 |     plt.plot(X,Y, linewidth=2)
103 |     plt.plot(X,Z,color="red", linewidth=2)
104 |     plt.xlabel("r",fontsize=40)
105 |     plt.ylabel("f(r)",fontsize=40)
106 |     plt.xticks(fontsize=30)
107 |     plt.yticks(fontsize=30)
108 |     plt.grid(True)
109 |     plt.tight_layout()
110 |     plt.show()
111 |
112 | print(valeurs_negatives(f,R))
113 | print(zone_tore(f,R,p_dicho))
114 | x1,x2=valeurs_negatives(f,R)
115 | x1p,x2p=zone_tore(f,R,p_dicho)
116 |
117 | graphe_f(x1p-10** $-7.5$ ,x2p+10** $-7.5$ ,p_dicho)
118 |
119 | etude_de_f=plt.figure("Etude de f",figsize = (16, 9))
120 | plt.gcf().subplots_adjust(left = 0.25, bottom = 0.1,right = 0.95, top = 0.93, wspace = 0.4, hspace = 0.4)
121 |
122 | ax = etude_de_f.add_subplot(1, 2, 1) #Table
123 | data=[[R,"m"],
124 |       [rho,"m"],
125 |       [I,"A"],
126 |       [N,""],
127 |       [q,"C"],
128 |       [m,"kg"],
129 |       [B,""],
130 |       [v0,"m/s"],
131 |       [(r0,theta0,z0),"(m, rad, m)"],
132 |       ["",""],
133 |       [(x1,x2),""],
134 |       [dichotomie(f,x1,R,p_dicho),""],
135 |       [dichotomie(f,R,x2,p_dicho),""],
136 |       [p_dicho,""]
137 |       ]
138 | column_labels=["Valeur Numérique","Unité"]
139 | row_labels=["Grand rayon R","Petit rayon p","Intensité courant bobines I","Nombre de Bobines","Charge
particule q", "Masse particule m","Constante  $\beta$ ","Vitesse initiale de la particule","Coordonnées
initiales de la particule","","Intervalle d'étude","R1 (premier point d'annulation de f)","R2 (deuxième point
d'annulation de f)","Précision"]
140 | plt.axis('tight')
141 | plt.axis('off')
142 | plt.table(cellText=data,colLabels=column_labels,rowLabels=row_labels,loc="center")
143 |
144 | ax = etude_de_f.add_subplot(1, 2, 2)
145 | X=np.arange(x1,x2,p_graphe)
146 | Y=[]
147 | for i in X:
148 |     Y.append(f(i))
149 | plt.plot(X,Y)
150 | plt.grid(True)
151 | plt.xlabel("r",fontsize=30)
152 | plt.ylabel("f(r)",fontsize=30)
153 | plt.xticks(fontsize=20)
154 | plt.yticks(fontsize=20)
155 | plt.show()
156 |
157 | ## Résolution du Système:
158 |
159 | ## Fonctions utiles:
160 |
161 | def est_dans_tore(r,z,p,R):
162 |     #Vérifie si le point de coord. (r,z) est dans le tore de grand rayon R et de petit rayon p
163 |     if (r-R)**2+z**2 <= p**2:
164 |         return True
165 |     else:
166 |         return False
167 |
168 | def mult_pol_cart(r,theta):

```

```

169|     #converti coordonnées cylindriques en coordonnées cartésiennes
170|     x=r*cos(theta)
171|     y=r*sin(theta)
172|     return(x,y) # Faible précision
173|
174| def mult_l_pol_cart(R_list,THETA,Z,T):
175|     #Prend en argument les listes utiles pour la méthode d'euler et renvoie 3 listes contenant les
coordonnées cartésiennes (liste 1 -> x, liste 2 -> y, liste 3 -> z)
176|     X,Y=[],[]
177|     for i in range(len(T)):
178|         x,y=mult_pol_cart(R_list[i],THETA[i])
179|         X.append(x)
180|         Y.append(y)
181|     return(X,Y,Z)
182|
183| ## Méthode d'Euler:
184|
185| R_Euler,S_Euler,THETA_Euler,Z_Euler,T_Euler=[r0],[v0],[theta0],[z0],[t0] #Initialisation des listes de
valeur
186|
187| r,s,theta,z,t=r0,v0,theta0,z0,t0 #Initialisation des variables
188|
189| def f_s(r):
190|     return (((R*v0)**2)/r**3)-(B**2)*(log(r/R)/r)
191|
192| def f_theta(r):
193|     return (R*v0)/(r**2)
194|
195| def f_z(r):
196|     return B*log(r/R)
197|
198| i=0
199|
200| while (t<=Tmax) and (est_dans_tore(r,z,rho,R)):
201|     print(est_dans_tore(r,z,rho,R))
202|     r, s, theta, z, t = r+h*s, s+h*f_s(r), theta+h*f_theta(r), z+h*f_z(r),t+h
203|     i+=1
204|     print(color.BOLD + str(i) + color.END,"\n",r,s,theta,z,t,"\n")
205|     R_Euler.append(r)
206|     S_Euler.append(s)
207|     THETA_Euler.append(theta)
208|     Z_Euler.append(z)
209|     T_Euler.append(t)
210|
211| ## Visualisation des résultats obtenus avec la méthode d'Euler:
212|
213| ## r(t):
214| plt.figure()
215| plt.plot(T_Euler,R_Euler,color="green")
216| plt.xlabel("t",fontsize=30)
217| plt.ylabel("r(t)",fontsize=30)
218| plt.xticks(fontsize=20)
219| plt.yticks(fontsize=20)
220| plt.tight_layout()
221| plt.show()
222|
223| ## z(t):
224| plt.figure()
225| plt.plot(T_Euler,Z_Euler,color="green")
226| plt.xlabel("t",fontsize=30)
227| plt.ylabel("z(t)",fontsize=30)
228| plt.xticks(fontsize=20)
229| plt.yticks(fontsize=20)
230| plt.tight_layout()
231| plt.show()
232|
233| ## Général:
234| X,Y,Z=mult_l_pol_cart(R_Euler,THETA_Euler,Z_Euler,T_Euler)
235|
236| euler1=plt.figure("Méthode d'Euler ",figsize = (16, 9))
237| plt.gcf().subplots_adjust(left = 0.1, bottom = 0.1,right = 0.98, top = 0.98, wspace = 0.4, hspace = 0.4)
238|
239| ax = euler1.add_subplot(3, 3, 1) #Table
240| data=[[R,"m"],
241|       [rho,"m"],
242|       [I,"A"],
243|       [N,""],
244|       [q,"C"],
245|       [m,"kg"],
246|       [B,""],
247|       [v0,"m/s"],
248|       [(r0,theta0,z0),"(m, rad, m)"],
249|       [Tmax,""],
250|       [h,""],
251|       [i,"étapes"],
252|       [i*h,"s"]
253|       ]

```



```

254| column_labels=["Valeur Numérique","Unité"]
255| row_labels=["Grand rayon R","Petit rayon p","Intensité courant bobines I","Nombre de Bobines","Charge
particule q", "Masse particule m",r"Constante  $\beta$ ", "Vitesse initiale particule", "Coordonnée initiales de la
particule", "Intervalle de temps maximal", "Pas utilisé dans la méthode d'Euler", "Collision au bout
de", "Collision au bout de"]
256| plt.axis('tight')
257| plt.axis('off')
258| plt.table(cellText=data,colLabels=column_labels,rowLabels=row_labels,loc="center")
259|
260| ax = euler1.add_subplot(3, 3, 2)
261| plt.plot(T_Euler,R_Euler,label="r en fonction de t",color="green")
262| plt.legend()
263|
264| ax = euler1.add_subplot(3, 3, 3)
265| plt.plot(T_Euler,S_Euler,label="s en fonction de t",color="green")
266| plt.legend()
267|
268| ax = euler1.add_subplot(3, 3, 4) #Portrait de phase
269| plt.plot(R_Euler,S_Euler,label="Portrait de phase",color="lime")
270| plt.legend()
271|
272| ax = euler1.add_subplot(3, 3, 5)
273| plt.plot(T_Euler,THETA_Euler,label=r"$\theta$ en fonction de t",color="green")
274| plt.legend()
275|
276| ax=euler1.add_subplot(3,3,6)
277| plt.plot(T_Euler,Z_Euler,label="z en fonction de t",color="green")
278| plt.legend()
279|
280| ax=euler1.add_subplot(3,3,7, projection='3d') #Trajectoire
281|
282| ax.plot(X, Y, Z, label='Courbe',color="green")
283| plt.title("Trajectoire d'un Électron")
284| ax.set_xlabel('x')
285| ax.set_ylabel('y')
286| ax.set_zlabel('z')
287| plt.tight_layout()
288|
289| ax=euler1.add_subplot(3,3,8, projection='3d') #Tore
290| (theta, phi) = np.meshgrid(np.linspace(0, 2 * np.pi, 41),
291|                             np.linspace(0, 2 * np.pi, 41))
292|
293| x = (1.9 + np.cos(phi)) * np.cos(theta)
294| y = (1.9 + np.cos(phi)) * np.sin(theta)
295| z = 0.7*np.sin(phi)
296|
297| dplot = ax.plot_surface(x,y,z,color='whitesmoke',alpha=0.5)
298| ax.set_xlabel='x',
299|       ylabel='y',
300|       zlabel='z',
301|       xlim = [-3.2, 3.2],
302|       ylim = [-3.2, 3.2],
303|       zlim = [-1, 1],
304|       xticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
305|       yticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
306|       zticks = [-0.7, 0, 0.7],
307|       title='Tore')
308| for t in ax.xaxis.get_major_ticks(): t.label.set_fontsize(5)
309| for t in ax.yaxis.get_major_ticks(): t.label.set_fontsize(5)
310| for t in ax.zaxis.get_major_ticks(): t.label.set_fontsize(5)
311| ax.scatter(X, Y, Z, marker='p',color="green") # Tracé des points 3D
312|
313| ax=euler1.add_subplot(3,3,9, projection='3d')
314| ax.scatter(X, Y, Z, marker='d',color="green") # Tracé des points 3D
315| plt.title("Positions de la particule")
316| ax.set_xlabel('x')
317| ax.set_ylabel('y')
318| ax.set_zlabel('z')
319| plt.tight_layout()
320|
321| plt.show()
322|
323| ## Trajectoire:
324| traj = plt.figure()
325| X,Y,Z=mult_l_pol_cart(R_list,THETA,Z,T)
326| ax = traj.gca(projection='3d') # Affichage en 3D
327| ax.plot(X, Y, Z, label='Courbe') # Tracé de la courbe 3D
328| plt.title("Trajectoire d'un Électron")
329| ax.set_xlabel('x')
330| ax.set_ylabel('y')
331| ax.set_zlabel('z')
332| plt.tight_layout()
333| plt.show()
334|
335| ## Tore:
336| X,Y,Z=mult_l_pol_cart(R_Euler,THETA_Euler,Z_Euler,T_Euler)
337| tore = plt.figure("Tore",clear=True)

```

```

338| ax = tore.add_subplot(1, 1, 1, projection='3d')
339|
340| (theta, phi) = np.meshgrid(np.linspace(0, 2 * np.pi, 41),
341|                             np.linspace(0, 2 * np.pi, 41))
342|
343| x = (1.9 + np.cos(phi)) * np.cos(theta)
344| y = (1.9 + np.cos(phi)) * np.sin(theta)
345| z = 0.7*np.sin(phi)
346|
347| dplot = ax.plot_surface(x,y,z,color='whitesmoke' ,alpha=0.5)
348| ax.set(xlabel='x',
349|        ylabel='y',
350|        zlabel='z',
351|        xlim = [-3.2, 3.2],
352|        ylim = [-3.2, 3.2],
353|        zlim = [-1, 1],
354|        xticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
355|        yticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
356|        zticks = [-0.7, 0, 0.7],
357|        title='Tore')
358| ax.scatter(X, Y, Z, marker='p')
359| tore.tight_layout()
360| plt.show()
361| ##
362| for angle in range(0, 360):
363|     ax.view_init(angle, angle)
364|     plt.draw()
365|     plt.pause(.001)
366|
367| ## Méthode de Heun:
368|
369| def f_s(r):
370|     return ((R*v0)**2)/r**3 - (B**2)*(log(r/R)/r)
371|
372| def f_theta(r):
373|     return (R*v0)/(r**2)
374|
375| def f_z(r):
376|     return B*log(r/R)
377|
378| r1,s1,theta1,z1=r0,v0,theta0,z0 #Initialisation des variables
379|
380| r,s,theta,z,t=r0,v0,theta0,z0,t0 #Initialisation des variables
381|
382| R_Heun,S_Heun,THETA_Heun,Z_Heun,T_Heun=[r0],[v0],[theta0],[z0],[t0] #Initialisation des listes de valeur
383|
384| i=0
385|
386| while (t<=Tmax) and (est_dans_tore(r,z,rho,R)):
387|     #Méthode d'Euler pour approximation:
388|     print(est_dans_tore(r,z,rho,R))
389|     r1, s1, theta1, z1= r1+h*s1, s1+h*f_s(r1), theta1+h*f_theta(r1), z1+h*f_z(r1)
390|
391|     #Méthode de Heun:
392|     r, s, theta, z, t = r+h*(s+s1)/2, s+h*(f_s(r)+f_s(r1))/2, theta+h*(f_theta(r)+f_theta(r1))/2,
393| z+h*(f_z(r)+f_z(r1))/2, t+h
394|     i+=1
395|     print(i, "\n", r,s,theta,z,t, "\n")
396|     R_Heun.append(r)
397|     S_Heun.append(s)
398|     THETA_Heun.append(theta)
399|     Z_Heun.append(z)
400|     T_Heun.append(t)
401|
402| ## Visualisation des résultats obtenus avec la méthode de Heun:
403|
404| ## r(t):
405| plt.figure()
406| plt.plot(T_Heun,R_Heun,color="darkorange")
407| plt.xlabel("t",fontsize=30)
408| plt.ylabel("r(t)",fontsize=30)
409| plt.xticks(fontsize=20)
410| plt.yticks(fontsize=20)
411| plt.tight_layout()
412| plt.show()
413|
414| ## z(t):
415| plt.figure()
416| plt.plot(T_Heun,Z_Heun,color="darkorange")
417| plt.xlabel("t",fontsize=30)
418| plt.ylabel("z(t)",fontsize=30)
419| plt.xticks(fontsize=20)
420| plt.yticks(fontsize=20)
421| plt.tight_layout()
422| plt.show()
423| ## Comparasion Euler/Heun -> r(t):

```

```

424| plt.figure()
425| plt.plot(T_Euler,R_Euler,label="Euler",color="green")
426| plt.plot(T_Heun,R_Heun,"--",label="Heun",color="darkorange")
427| plt.xlabel("t",fontsize=30)
428| plt.ylabel("r(t)",fontsize=30)
429| plt.xticks(fontsize=20)
430| plt.yticks(fontsize=20)
431| plt.legend(loc="right")
432| plt.tight_layout()
433| plt.show()
434|
435| ## Comparasion Euler/Heun -> z(t):
436| plt.figure()
437| plt.plot(T_Euler,Z_Euler,label="Euler",color="green")
438| plt.plot(T_Heun,Z_Heun,"--",label="Heun",color="darkorange")
439| plt.xlabel("t",fontsize=30)
440| plt.ylabel("z(t)",fontsize=30)
441| plt.xticks(fontsize=20)
442| plt.yticks(fontsize=20)
443| plt.tight_layout()
444| plt.legend(loc="right")
445| plt.show()
446|
447| ## Comparasion Euler/Heun -> r(t) (zoom):
448| i=4350
449| j=4450
450| echantillon_Euler=R_Euler[i:j]
451| echantillon_Euler_T=T_Euler[i:j]
452| echantillon_Heun=R_Heun[i:j]
453| echantillon_Heun_T=T_Heun[i:j]
454|
455| plt.figure()
456| plt.plot(echantillon_Euler_T,echantillon_Euler,label="Euler",color="green")
457| plt.plot(echantillon_Heun_T,echantillon_Heun,label="Heun",color="darkorange")
458| plt.xlabel("t",fontsize=30)
459| plt.ylabel("r(t)",fontsize=30)
460| plt.xticks(fontsize=20)
461| plt.yticks(fontsize=20)
462| plt.legend(loc="right")
463| plt.tight_layout()
464| plt.show()
465|
466| ## Général:
467| X,Y,Z=mult_l_pol_cart(R_Heun,THETA_Heun,Z_Heun,T_Heun)
468|
469| heun=plt.figure("Méthode de Heun",figsize = (16, 9))
470| #plt.title("Méthode d'Euler 1")
471| plt.gcf().subplots_adjust(left = 0.1, bottom = 0.1,right = 0.98, top = 0.98, wspace = 0.4, hspace = 0.4)
472|
473|
474| ax = heun.add_subplot(3, 3, 1) #Table
475| data=[[R,"m"],
476|       [rho,"m"],
477|       [I,"A"],
478|       [N,""],
479|       [q,"C"],
480|       [m,"kg"],
481|       [B,""],
482|       [v0,"m/s"],
483|       [(r0,theta0,z0),"(m, rad, m)"],
484|       [Tmax,""],
485|       [h,""],
486|       [i,"étapes"],
487|       [i*h,"s"]
488|       ]
489| column_labels=["Valeur Numérique","Unité"]
490| row_labels=["Grand rayon R","Petit rayon p","Intensité courant bobines I","Nombre de Bobines","Charge
particule q", "Masse particule m","Constante  $\beta$ ","Vitesse initiale particule","Coordonnée initiales
particules","Intervalle de temps maximal","Pas utilisé dans la méthode d'Euler","Collision au bout
de","Collision au bout de"]
491| plt.axis('tight')
492| plt.axis('off')
493| plt.table(cellText=data,colLabels=column_labels,rowLabels=row_labels,loc="center")
494|
495| ax = heun.add_subplot(3, 3, 2)
496| plt.plot(T_Heun,R_Heun,label="r en fonction de t",color="darkorange")
497| plt.legend()
498|
499| ax = heun.add_subplot(3, 3, 3)
500| plt.plot(T_Heun,S_Heun,label="s en fonction de t",color="darkorange")
501| plt.legend()
502|
503| ax = heun.add_subplot(3, 3, 4) #Portrait de phase
504| plt.plot(R_Heun,S_Heun,label="Portrait de phase",color="yellow")
505| plt.legend()
506|
507| ax = heun.add_subplot(3, 3, 5)

```

```

508 plt.plot(T_Heun,THETA_Heun,label=r"$\theta$ en fonction de t",color="darkorange")
509 plt.legend()
510
511 ax=heun.add_subplot(3,3,6)
512 plt.plot(T_Heun,Z_Heun,label="z en fonction de t",color="darkorange")
513 plt.legend()
514
515 ax=heun.add_subplot(3,3,7, projection='3d') #Trajectoire
516
517 ax.plot(X, Y, Z, label='Courbe',color="darkorange")
518 plt.title("Trajectoire d'un Électron")
519 ax.set_xlabel('x')
520 ax.set_ylabel('y')
521 ax.set_zlabel('z')
522 plt.tight_layout()
523
524 ax=heun.add_subplot(3,3,8, projection='3d') #Tore
525 (theta, phi) = np.meshgrid(np.linspace(0, 2 * np.pi, 41),
526                             np.linspace(0, 2 * np.pi, 41))
527
528 x = (1.9 + np.cos(phi)) * np.cos(theta)
529 y = (1.9 + np.cos(phi)) * np.sin(theta)
530 z = 0.7*np.sin(phi)
531
532 dplot = ax.plot_surface(x,y,z,color='whitesmoke',alpha=0.5)
533 ax.set(xlabel='x',
534        ylabel='y',
535        zlabel='z',
536        xlim = [-3.2, 3.2],
537        ylim = [-3.2, 3.2],
538        zlim = [-1, 1],
539        xticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
540        yticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
541        zticks = [-0.7, 0, 0.7],
542        title='Tore')
543 for t in ax.xaxis.get_major_ticks(): t.label.set_fontsize(5)
544 for t in ax.yaxis.get_major_ticks(): t.label.set_fontsize(5)
545 for t in ax.zaxis.get_major_ticks(): t.label.set_fontsize(5)
546 ax.scatter(X, Y, Z, marker='p',color="darkorange") # Tracé des points 3D
547
548 ax=heun.add_subplot(3,3,9, projection='3d')
549 ax.scatter(X, Y, Z, marker='d',color="darkorange") # Tracé des points 3D
550 plt.title("Positions de la particule")
551 ax.set_xlabel('x')
552 ax.set_ylabel('y')
553 ax.set_zlabel('z')
554 plt.tight_layout()
555
556 plt.show()
557
558 ## Méthode utilisant le module Scipy:
559
560 def instant_collision(dans_tore):
561     i=0
562     intervalle_temp=T_calcul/nbr_points
563     for i in range(len(dans_tore)):
564         if dans_tore[i]<=0:
565             return(i,i*intervalle_temp)
566     return("pas de sortie de tore","pas de sortie de tore")
567
568
569 def f_s(r):
570     return (((R*v0)**2)/r**3)-(B**2)*(log(r/R)/r)
571
572 def f_theta(r):
573     return (R*v0)/(r**2)
574
575 def f_z(r):
576     return B*log(r/R)
577
578 def dY(Y,t):
579     r,s,theta,z=Y
580     dYdt=[s,f_s(r),f_theta(r),f_z(r)]
581     return dYdt
582
583 Y0=[r0,v0,theta0,z0] #CI
584
585 T_calcul=10**-10
586 nbr_points=10000
587 t=np.linspace(0,T_calcul,nbr_points)
588
589 from scipy.integrate import odeint
590 sol = odeint(dY,Y0,t)
591 #sol = odeint(dY,Y0,t,mxstep=50000000)
592
593 dans_tore=[rho**2 -(sol[i,0]-R)**2-sol[i,3]**2 for i in range(len(sol))]
594 Indice_collision,T_collision=instant_collision(dans_tore)

```

```

595| print(Indice_collision,T_collision)
596|
597| ## Visualisation des résultats obtenus avec la méthode Scipy:
598|
599| ## Général:
600| solg=plt.figure("Méthode scipy",figsize = (16, 9))
601| plt.gcf().subplots_adjust(left = 0.13, bottom = 0.1,right = 0.98, top = 0.95, wspace = 0.2, hspace = 0.4)
602|
603| ax = solg.add_subplot(3, 3, 1) #Table
604| data=[[R,"m"],
605|       [rho,"m"],
606|       [I,"A"],
607|       [N,""],
608|       [q,"C"],
609|       [m,"kg"],
610|       [B,""],
611|       [v0,"m/s"],
612|       [(r0,theta0,z0),"(m, rad, m)"],
613|       [T_calcul,"s"],
614|       [T_collision,"s"]
615|       ]
616| column_labels=["Valeur Numérique","Unité"]
617| row_labels=["Grand rayon R","Petit rayon p","Intensité courant bobines I","Nombre de Bobines","Charge
particule q", "Masse particule m",r"Constante $\beta$", "Vitesse initiale particule","Coordonnée initiales
particules","Intervalle de temps de calcul","Sortie de tore"]
618| plt.axis('tight')
619| plt.axis('off')
620| plt.table(cellText=data,colLabels=column_labels,rowLabels=row_labels,loc="center")
621|
622| ax = solg.add_subplot(3, 3, 2)
623| plt.plot(t,sol[:,0],label="r en fonction de t")
624| plt.legend()
625|
626| ax = solg.add_subplot(3, 3, 3)
627| plt.plot(t,sol[:,1],label="s en fonction de t")
628| plt.legend()
629|
630| ax = solg.add_subplot(3, 3, 4) #Portrait de phase
631| plt.plot(sol[:,0],sol[:,1],label="Portrait de phase",color="lightblue")
632| plt.legend()
633|
634| ax = solg.add_subplot(3, 3, 5)
635| plt.plot(t,sol[:,2],label=r"$\theta$ en fonction de t")
636| plt.legend()
637|
638| ax=solg.add_subplot(3,3,6)
639| plt.plot(t,sol[:,3],label="z en fonction de t")
640| plt.legend()
641|
642| ax=solg.add_subplot(3,3,8)
643| plt.plot(t,dans_tore,label="positif<=>particule dans le tore")
644| plt.yticks([0])
645| plt.grid()
646|
647| plt.show()
648|
649| ## r(t):
650| plt.figure()
651| plt.plot(t, sol[:, 0], label='r(t)')
652| plt.legend(loc='best')
653| plt.xlabel('t',fontsize=30)
654| plt.ylabel('r(t)',fontsize=30)
655| plt.xticks(fontsize=20)
656| plt.yticks(fontsize=20)
657| plt.tight_layout()
658| plt.show()
659|
660| ## r(t) (avec horizontales):
661| x1p,x2p=zone_tore(f,R,p_dicho)
662| print(x1p,x2p)
663|
664| plt.figure()
665| Y1=[x1p for i in range(len(sol[:, 0]))]
666| Y2=[x2p for i in range(len(sol[:, 0]))]
667| plt.plot(t, sol[:, 0], label='r(t)')
668| plt.plot(t,Y1,"red")
669| plt.plot(t,Y2,"red")
670| plt.legend(loc='best')
671| plt.xlabel('t',fontsize=30)
672| plt.ylabel('r(t)',fontsize=30)
673| plt.xticks(fontsize=20)
674| plt.yticks(fontsize=20)
675| plt.tight_layout()
676| plt.show()
677|
678| ## z(t):
679| plt.figure()

```

```

680| plt.plot(t, sol[:, 3], label='z(t)')
681| plt.legend(loc='best')
682| plt.xlabel('t',fontsize=30)
683| plt.ylabel('z(t)',fontsize=30)
684| plt.xticks(fontsize=20)
685| plt.yticks(fontsize=20)
686| plt.tight_layout()
687| plt.show()
688|
689| ## Comparasion méthode Euler/Scipy:
690| plt.figure()
691| plt.plot(t, sol[:, 0], label='Scipy',linewidth=2.2)
692| plt.plot(T_Euler,R_Euler,label="Euler",linewidth=1, color="green")
693| plt.legend(loc='best')
694| plt.xlim(0,0.6*10**-10)
695| plt.xlabel('t',fontsize=30)
696| plt.ylabel('r(t)',fontsize=30)
697| plt.xticks(fontsize=20)
698| plt.yticks(fontsize=20)
699| plt.tight_layout()
700| plt.show()
701|
702| ## Négatif <=> Sortie du tore:
703|
704| plt.figure()
705| tore=[rho**2 -(sol[i,0]-R)**2-sol[i,3]**2 for i in range(len(sol))]
706|
707| plt.plot(t,tore)
708| plt.xticks(fontsize=20)
709| plt.yticks([0],fontsize=20)
710| plt.grid()
711| plt.tight_layout()
712| plt.show()
713|
714| ##Négatif <=> Sortie du tore (réduit):
715| plt.figure()
716| plt.plot(t,dans_tore,label="positif<=>particule dans le tore")
717| plt.plot(t,[0 for i in range(len(t))],color="red")
718| plt.plot([T_collision for i in range(10)],[i-1 for i in range(10)],color="red")
719| plt.yticks([0],fontsize=20)
720| plt.xticks([0,5.4*10**-11,6*10**-11],fontsize=20)
721| #plt.xticks(fontsize=20)
722| plt.xlim(0,6*10**-11)
723| plt.ylim(-0.1,0.5)
724| plt.grid()
725| plt.tight_layout()
726| plt.show()
727|
728| ##Tore:
729|
730| def pol_cart(list):
731|     r,theta,z = list[0],list[2],list[3]
732|     x=r*cos(theta)
733|     y=r*sin(theta)
734|     return(x,y,z)
735|
736| def l_pol_cart(sol):
737|     X,Y,Z=[],[],[]
738|     for i in range(len(sol)):
739|         x,y,z=pol_cart(sol[i])
740|         X.append(x)
741|         Y.append(y)
742|         Z.append(z)
743|     return(X,Y,Z)
744|
745| X,Y,Z=l_pol_cart(sol[:540])
746| tore = plt.figure("Tore",clear=True)
747| ax = tore.add_subplot(1, 1, 1, projection='3d')
748|
749| (theta, phi) = np.meshgrid(np.linspace(0, 2 * np.pi, 41),
750|                             np.linspace(0, 2 * np.pi, 41))
751|
752| x = (1.9 + np.cos(phi)) * np.cos(theta)
753| y = (1.9 + np.cos(phi)) * np.sin(theta)
754| z = 0.7*np.sin(phi)
755|
756| dplot = ax.plot_surface(x,y,z,color='whitesmoke',alpha=0.5)
757| ax.set(xlabel='x',
758|        ylabel='y',
759|        zlabel='z',
760|        xlim = [-3.2, 3.2],
761|        ylim = [-3.2, 3.2],
762|        zlim = [-1, 1],
763|        xticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
764|        yticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
765|        zticks = [-0.7, 0, 0.7],
766|        title='Tore')

```

```
767| ax.scatter(X, Y, Z, marker='p')
768| tope.tight_layout()
769|
770| plt.show()
771|
772| ##
773| for angle in range(0, 360):
774|     ax.view_init(10, angle)
775|     plt.draw()
776|     plt.pause(.001)
```

```

001 | ## Simulation Principale:
002 |
003 | ## Initialisation:
004 |
005 | import matplotlib.pyplot as plt
006 | from math import *
007 | import numpy as np
008 | from mpl_toolkits.mplot3d import axes3d
009 | import copy as c
010 |
011 | R=2.40 #Grand rayon du tore
012 | rho=0.7 #Rayon du tore
013 | I=1400 #Intensité du courant dans les bobines
014 | N=18*2028 #Nombre de spires
015 | q=1.6*(10)**-19 #Charge élémentaire
016 | mu=4*pi*10**-7 #Perméabilité du vide
017 | m=9.1*10**-31 #Masse e-
018 | B=(q*mu*N*I)/(2*pi*m)
019 |
020 | #v0=1.3*10**4 #Conservation de la quantité de mouvement
021 | #v0=10**11
022 | #v0=1.2*10**5
023 | #v0=8*10**7 #D'après Thermodynamique (T=150*10**6K)
024 | v0=10**6
025 |
026 | t0=0
027 | r0=R
028 | theta0=0
029 | z0=0
030 |
031 | p_dicho=10**-10
032 |
033 | T_calcul=5.4*10**-11
034 | nbr_points=10000
035 | t=np.linspace(0,T_calcul,nbr_points)
036 | #maxstep=50000000
037 | maxstep=10
038 | #maxstep=1000000 # Maximum number of (internally defined) steps allowed for each integration point in t.
039 |
040 | Racine = "F:/TIPE/Info/"
041 | FileName = str(v0)+"_"+str(T_calcul)+"_"+str(nbr_points)+"_"+str(maxstep)
042 |
043 | def FileWriteCsv(file,array):
044 |     for i in range(array.shape[0]):
045 |         for j in range(array.shape[1]):
046 |             file.write(str(array[i,j]))
047 |             if (j != array.shape[1]-1):
048 |                 file.write(",")
049 |             file.write("\n")
050 |
051 | def linsplt(l):
052 |     return list(map(float,l.split(",")))
053 |
054 | def FileReadCsv(file):
055 |     lines = file.readlines()
056 |     return np.array(list(map(linsplt,lines)))
057 |
058 | def f(r):
059 |     #fonction f issue du syst.
060 |     return 2*(v0**2)-((R*v0)**2/(r**2))-(B*log(r/R))**2
061 |
062 | def dichotomie(f,r1,r2,p):
063 |     #Prend en argument une fonction f, un intervalle [r1,r2] et une précision et détermine le point où f
s'annule avec la précision p
064 |     while r2-r1>p:
065 |         m=(r1+r2)*0.5
066 |         if f(m)*f(r1)<0:
067 |             r2=m
068 |         else:
069 |             r1=m
070 |     return((r1+r2)*0.5)
071 |
072 | def valeurs_negatives(f,R):
073 |     #Prend en argument une fonction f et R et détermine deux valeurs à gauche et à droite de R pour
lesquelles la fonction f est négative
074 |     x1,x2=R,R
075 |     while f(x1)>0:
076 |         x1=x1/2
077 |     while f(x2)>0:
078 |         x2=2*x2
079 |     x1=x1/2
080 |     x2=2*x2
081 |     return(x1,x2)
082 |
083 | def zone_tore(f,R,p):
084 |     #prend en argument une fonction f, le grand rayon du tore R et une "précision" p et détermine
l'intervalle de r que peut atteindre la particule

```



```

085 |     x1i,x2i=valeurs_negatives(f,R)
086 |     x1=dichotomie(f,x1i,R,p)
087 |     x2=dichotomie(f,R,x2i,p)
088 |     return(x1,x2)
089 |
090 | def instant_collision(dans_tore):
091 |     i=0
092 |     intervalle_temp=T_calcul/nbr_points
093 |     for i in range(len(dans_tore)):
094 |         if dans_tore[i]<=0:
095 |             return(i,i*intervalle_temp)
096 |     return("pas de sortie de tore","pas de sortie de tore")
097 |
098 | ## Enregistrement:
099 |
100 | def f_s(r):
101 |     return (((R*v0)**2)/r**3)-(B**2)*(log(r/R)/r)
102 |
103 | def f_theta(r):
104 |     return (R*v0)/(r**2)
105 |
106 | def f_z(r):
107 |     return B*log(r/R)
108 |
109 | def dY(Y,t):
110 |     r,s,theta,z=Y
111 |     dYdt=[s,f_s(r),f_theta(r),f_z(r)]
112 |     return dYdt
113 |
114 | Y0=[r0,v0,theta0,z0] #CI
115 |
116 | from scipy.integrate import odeint
117 | sol = odeint(dY,Y0,t, mxstep=maxstep)
118 | #sol = odeint(dY,Y0,t)
119 |
120 | dans_tore=[rho**2 -(sol[i,0]-R)**2-sol[i,3]**2 for i in range(len(sol))]
121 | Indice_collision,T_collision=instant_collision(dans_tore)
122 | print(Indice_collision,T_collision)
123 |
124 | file = open(Racine+FileName+".csv","w")
125 | FileWriteCsv(file,sol)
126 | file.close()
127 |
128 | ## Lecture:
129 |
130 | #file = open(Racine+FileName+".csv","r")
131 | file = open(Racine+"1000000000_9.000000000000001e-09_10000_100000.csv","r")
132 | sol = FileReadCsv(file)
133 | file.close()
134 |
135 | dans_tore=[rho**2 -(sol[i,0]-R)**2-sol[i,3]**2 for i in range(len(sol))]
136 | Indice_collision,T_collision=instant_collision(dans_tore)
137 | print(Indice_collision,T_collision)
138 |
139 | ## Visualisation:
140 |
141 | ## Générale:
142 | solg=plt.figure("Méthode scipy",figsize = (16, 9))
143 | plt.gca().subplots_adjust(left = 0.13, bottom = 0.1, right = 0.98, top = 0.95, wspace = 0.2, hspace = 0.4)
144 |
145 | ax = solg.add_subplot(3, 3, 1) #Table
146 | data=[[R,"m"],
147 |       [rho,"m"],
148 |       [I,"A"],
149 |       [N,""],
150 |       [q,"C"],
151 |       [m,"kg"],
152 |       [B,""],
153 |       [v0,"m/s"],
154 |       [(r0,theta0,z0),"(m, rad, m)"],
155 |       [T_calcul,"s"],
156 |       [T_collision,"s"]
157 |       ]
158 | column_labels=["Valeur Numérique","Unité"]
159 | row_labels=["Grand rayon R","Petit rayon p","Intensité courant bobines I","Nombre de Bobines","Charge
particule q", "Masse particule m",r"Constante  $\beta$ ", "Vitesse initiale particule", "Coordonnée initiales
particules", "Intervalle de temps de calcul", "Sortie de tore"]
160 | plt.axis('tight')
161 | plt.axis('off')
162 | plt.table(cellText=data,colLabels=column_labels,rowLabels=row_labels,loc="center")
163 |
164 | ax = solg.add_subplot(3, 3, 2)
165 | plt.plot(t,sol[:,0],label="r en fonction de t")
166 | plt.legend()
167 |
168 | ax = solg.add_subplot(3, 3, 3)
169 | plt.plot(t,sol[:,1],label="s en fonction de t")

```

```

170| plt.legend()
171|
172| ax = solg.add_subplot(3, 3, 4) #Portrait de phase
173| plt.plot(sol[:,0],sol[:,1],label="Portrait de phase",color="lightblue")
174| plt.legend()
175|
176| ax = solg.add_subplot(3, 3, 5)
177| plt.plot(t,sol[:,2],label=r"$\theta$ en fonction de t")
178| plt.legend()
179|
180| ax=solg.add_subplot(3,3,6)
181| plt.plot(t,sol[:,3],label="z en fonction de t")
182| plt.legend()
183|
184| ax=solg.add_subplot(3,3,8)
185| plt.plot(t,dans_tore,label="positif<=>particule dans le tore")
186| plt.yticks([0])
187| plt.grid()
188|
189| plt.show()
190|
191| ## r(t):
192| plt.figure()
193| plt.plot(t, sol[:, 0], label='r(t)')
194| plt.legend(loc='best')
195| plt.xlabel('t',fontsize=30)
196| plt.ylabel('r(t)',fontsize=30)
197| plt.xticks(fontsize=20)
198| plt.yticks(fontsize=20)
199| #plt.xlim(0,10**-6)
200| plt.tight_layout()
201| plt.show()
202|
203| ## r(t) (avec horizontales):
204| x1p,x2p=zone_tore(f,R,p_dicho)
205| print(x1p,x2p)
206|
207| plt.figure()
208| Y1=[x1p for i in range(len(sol[:, 0]))]
209| Y2=[x2p for i in range(len(sol[:, 0]))]
210| plt.plot(t, sol[:, 0], label='r(t)')
211| plt.plot(t,Y1,"red")
212| plt.plot(t,Y2,"red")
213| plt.legend(loc='best')
214| plt.xlabel('t',fontsize=30)
215| plt.ylabel('r(t)',fontsize=30)
216| plt.xticks(fontsize=20)
217| plt.yticks(fontsize=20)
218| plt.tight_layout()
219| plt.show()
220|
221| ## z(t):
222| plt.figure()
223| plt.plot(t, sol[:, 3], label='z(t)')
224| plt.legend(loc='best')
225| plt.xlabel('t',fontsize=30)
226| plt.ylabel('z(t)',fontsize=30)
227| plt.xticks([0*10**-4,0.5*10**-4,10**-4],fontsize=20)
228| plt.yticks(fontsize=20)
229| plt.tight_layout()
230| plt.show()
231|
232| ## z(t) (réduit):
233| plt.figure()
234| plt.plot(t, sol[:, 3], label='z(t)')
235| plt.legend(loc='best')
236| plt.xlabel('t',fontsize=30)
237| plt.ylabel('z(t)',fontsize=30)
238| plt.xticks(fontsize=20)
239| plt.yticks(fontsize=20)
240| plt.xlim(0,10**-7)
241| plt.ylim(0,2.5*10**-5)
242| plt.tight_layout()
243| plt.show()
244|
245| ## Négatif <=> Sortie du tore:
246|
247| plt.figure()
248| tore=[rho**2 -(sol[i,0]-R)**2-sol[i,3]**2 for i in range(len(sol))]
249|
250| plt.plot(t,tore)
251| plt.xticks([0,5*10**-5,10**-4],fontsize=20)
252| plt.yticks(fontsize=20)
253| plt.grid()
254| plt.tight_layout()
255| plt.show()
256|

```

```

257| ##Négatif <=> Sortie du tore (réduit):
258| plt.figure()
259| plt.plot(t,dans_tore,label="positif<=>particule dans le tore")
260| plt.plot(t,[0 for i in range(len(t))],color="red")
261| plt.plot([T_collision for i in range(10)],[i-1 for i in range(10)],color="red")
262| plt.yticks([0],fontsize=20)
263| plt.xticks([0,5.4*10**-11,6*10**-11],fontsize=20)
264| #plt.xticks(fontsize=20)
265| plt.xlim(0,6*10**-11)
266| plt.ylim(-0.1,0.5)
267| plt.grid()
268| plt.tight_layout()
269| plt.show()
270|
271| ## Tore:
272|
273| def pol_cart(list):
274|     r,theta,z = list[0],list[2],list[3]
275|     x=r*cos(theta)
276|     y=r*sin(theta)
277|     return(x,y,z)
278|
279| def l_pol_cart(sol):
280|     X,Y,Z=[],[],[]
281|     for i in range(len(sol)):
282|         x,y,z=pol_cart(sol[i])
283|         X.append(x)
284|         Y.append(y)
285|         Z.append(z)
286|     return(X,Y,Z)
287|
288| X,Y,Z=l_pol_cart(sol)
289| tore = plt.figure("Tore",clear=True)
290| ax = tore.add_subplot(1, 1, 1, projection='3d')
291|
292| (theta, phi) = np.meshgrid(np.linspace(0, 2 * np.pi, 41),
293|                             np.linspace(0, 2 * np.pi, 41))
294|
295| x = (1.9 + np.cos(phi)) * np.cos(theta)
296| y = (1.9 + np.cos(phi)) * np.sin(theta)
297| z = 0.7*np.sin(phi)
298|
299| dplot = ax.plot_surface(x,y,z,color='whitesmoke',alpha=0.5)
300| ax.set(xlabel='x',
301|        ylabel='y',
302|        zlabel='z',
303|        xlim = [-3.2, 3.2],
304|        ylim = [-3.2, 3.2],
305|        zlim = [-1, 1],
306|        xticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
307|        yticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
308|        zticks = [-0.7, 0, 0.7],
309|        title='Tore')
310| ax.scatter(X, Y, Z, marker='p')
311| tore.tight_layout()
312| #ax.view_init(10, 45)
313| #ax.view_init(90,90)
314| ax.view_init(0,90)
315| plt.show()
316| ##
317| for angle in range(0, 360):
318|     ax.view_init(angle, angle)
319|     plt.draw()
320|     plt.pause(.001)

```