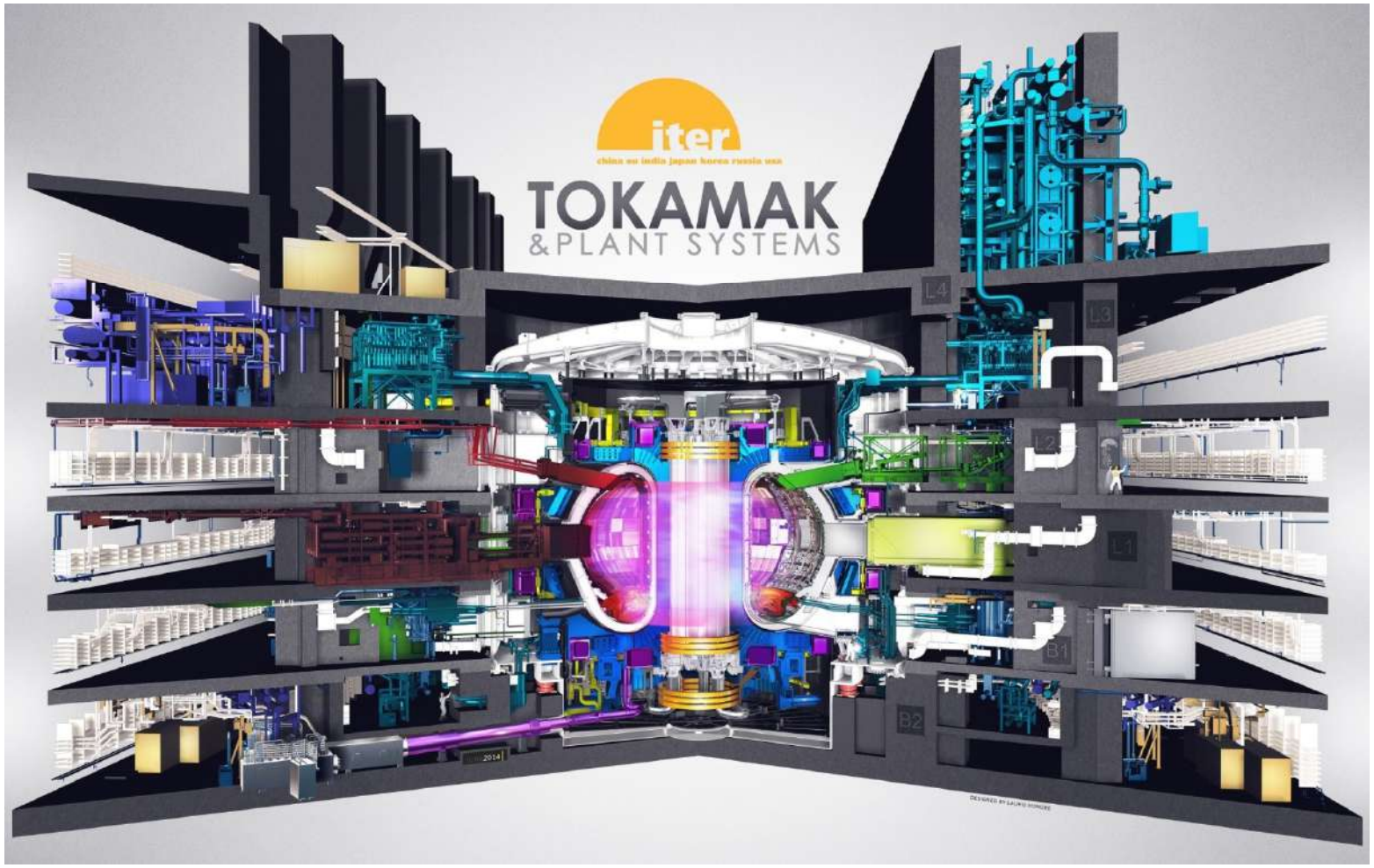
Fabien ALLEMAND

Candidat: 19403

Confinement magnétique de magma :

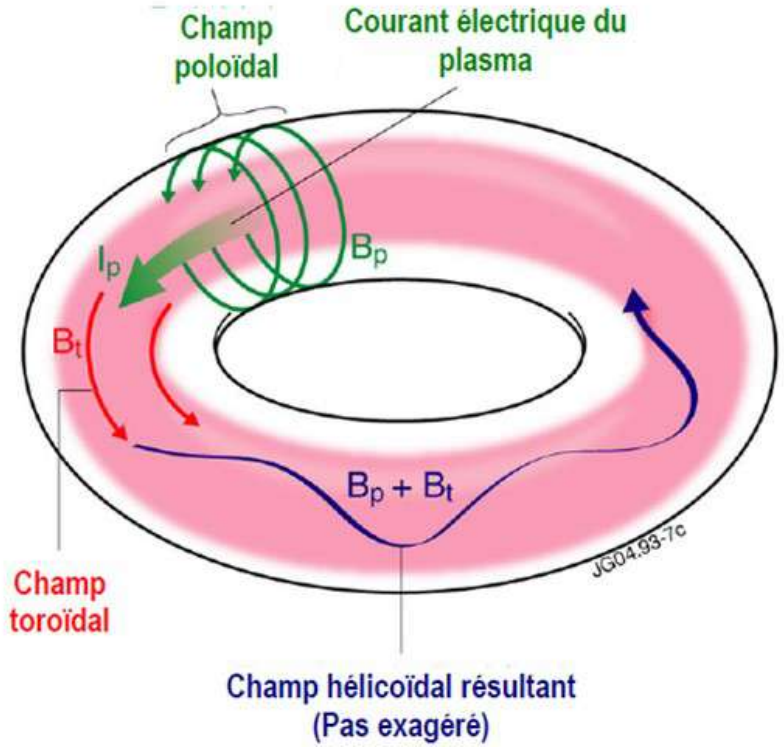
Un nouvel atout pour les énergies du futur



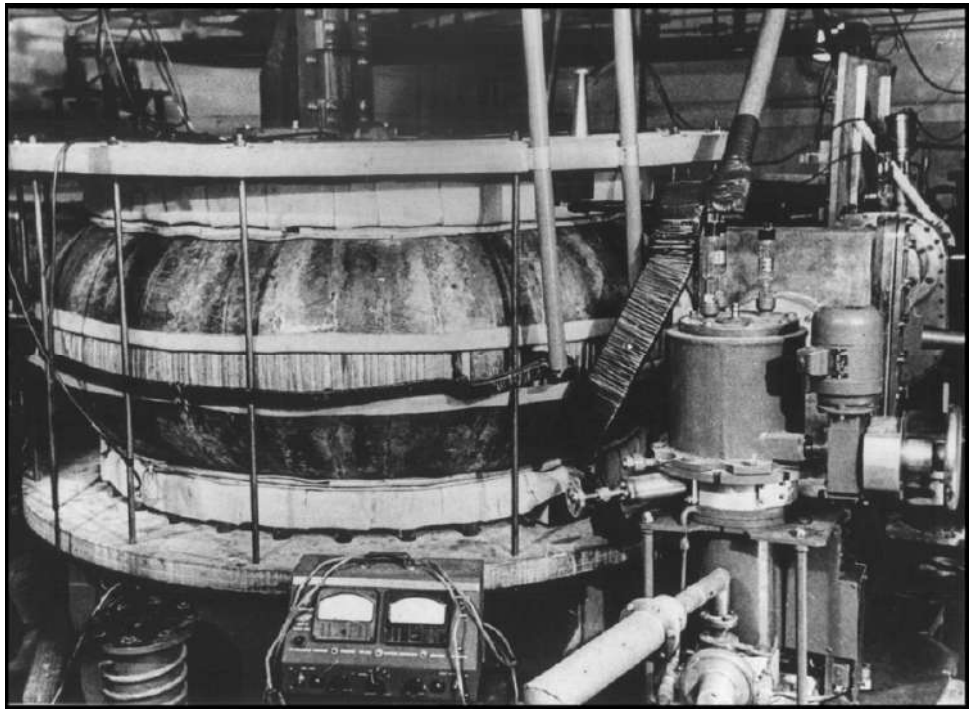
Source: iter.org

Tokamak

Champ magnétique à l'intérieur du tokamak



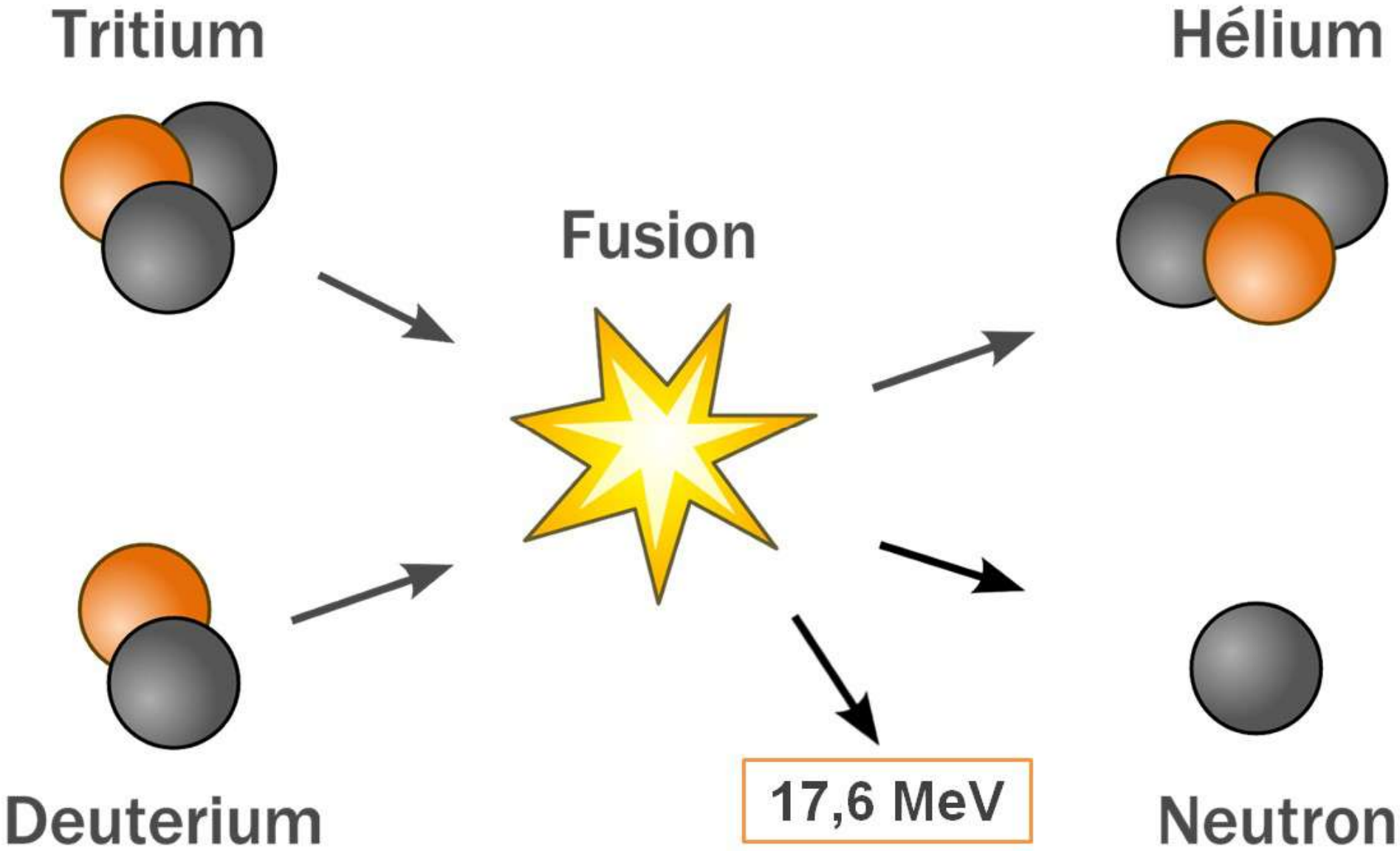
Tokamak T1 à l'institut Kourtchatov à Moscou



Issu du russe: ***to****roidalnaïa* ***ka****mera s* ***ma****gnitnymi* ***k****atouchkami,*
qui signifie: **chambre toroïdale avec bobines magnétiques**

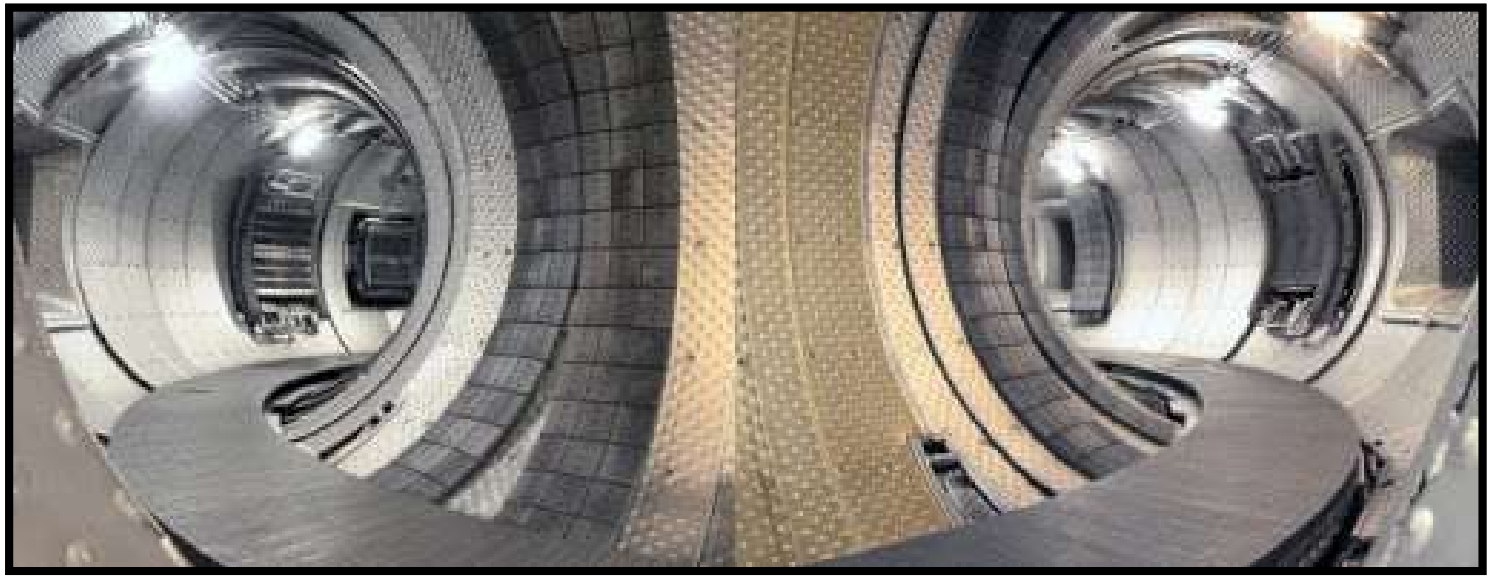
Source: iter.org

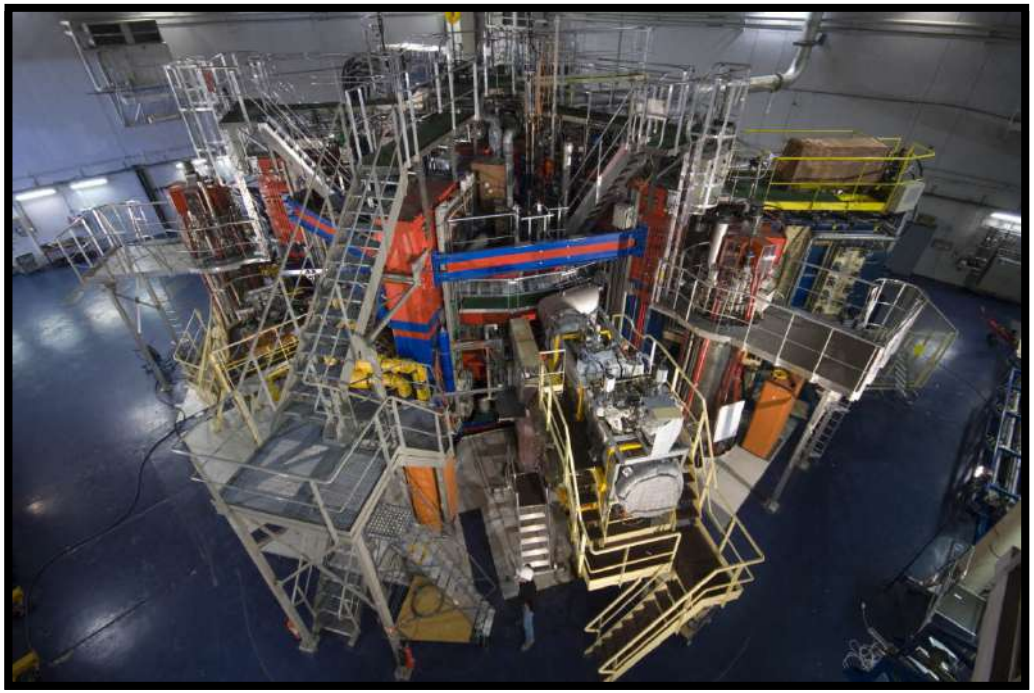
Réaction de fusion nucléaire



Source: medium.com

Tokamak Tore Supra

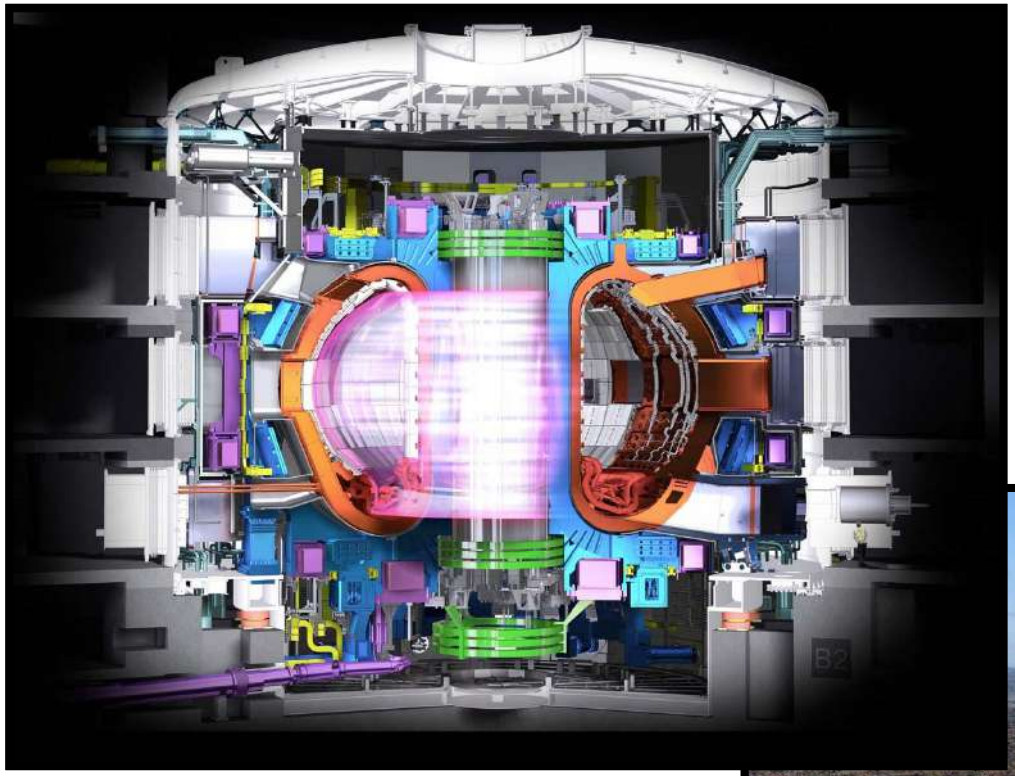




Source: cea.fr

Tokamak ITER

(International Thermonuclear Experimental Reactor)



iter

eu india japan korea russi

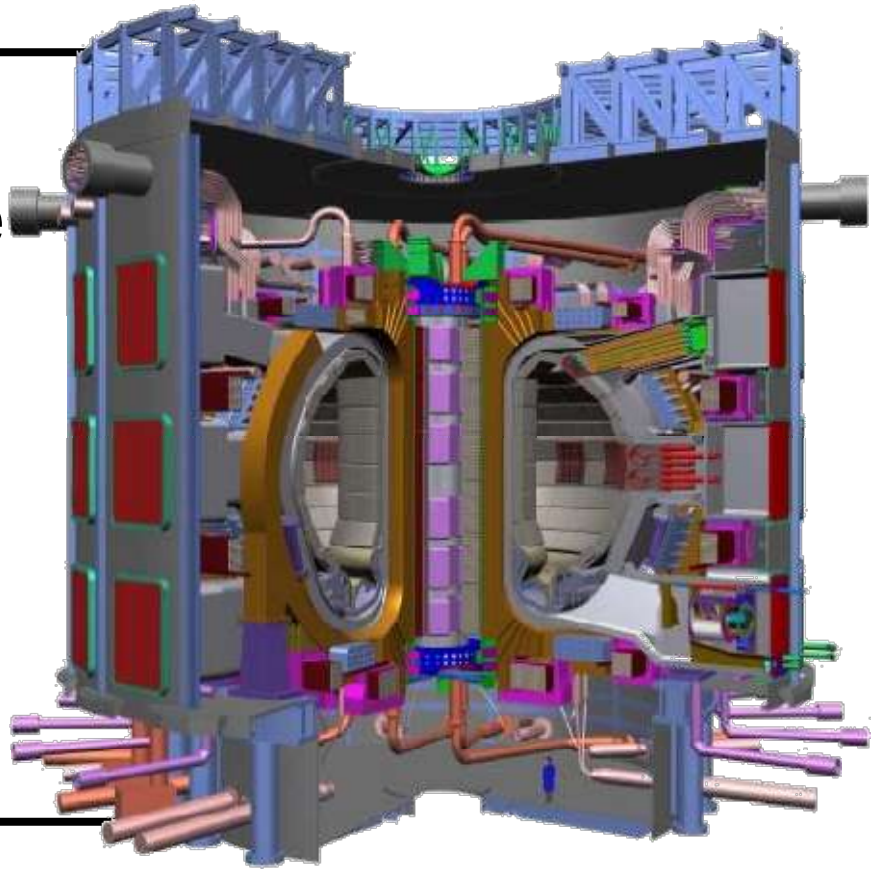
Source: iter.org

Plan:

I: Confinement Magnétique

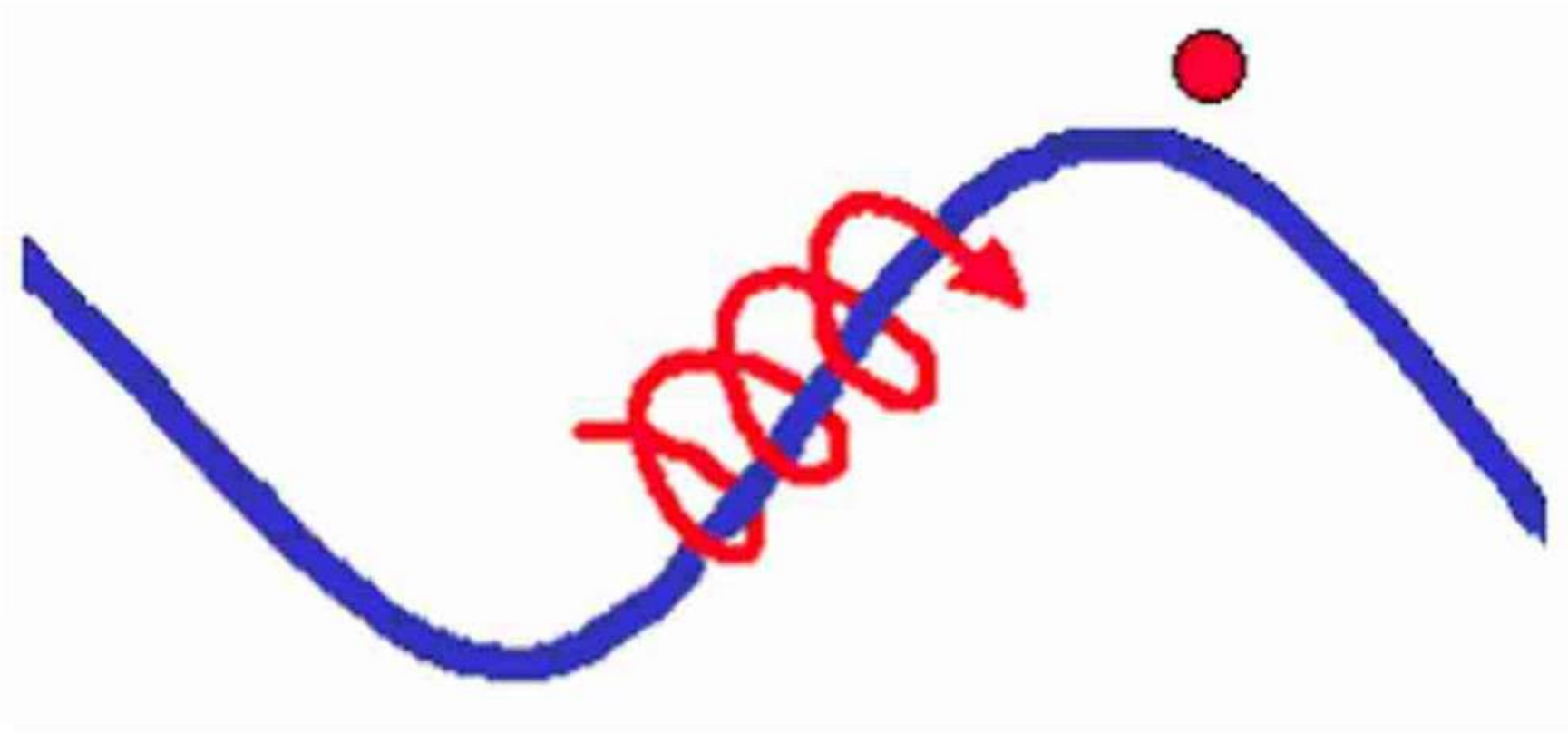
II: Etude d'un Tokamak

III: Amélioration du Modèle

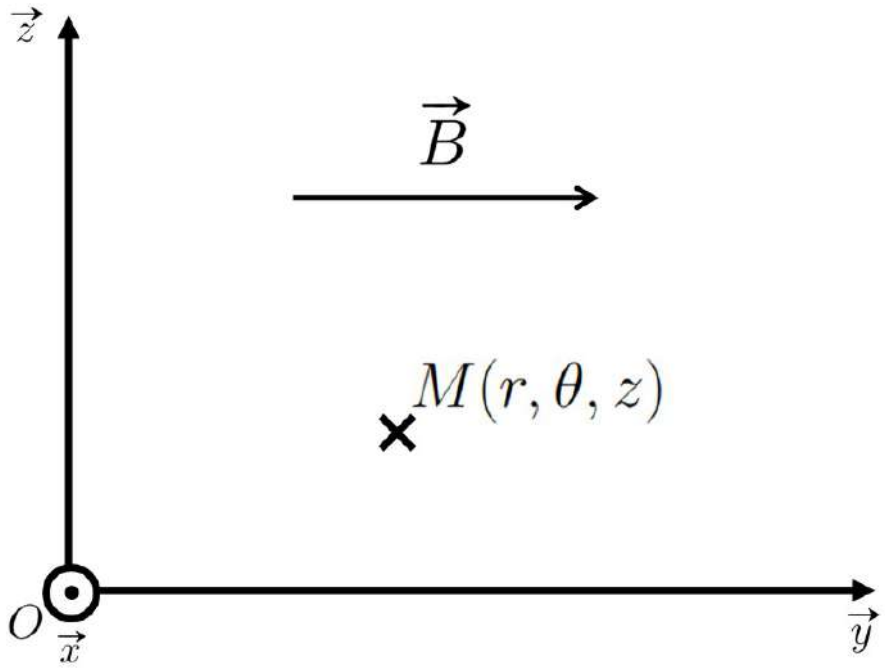


Source: www.fusion-magnetique.cea.fr

I: Confinement Magnétique



Etude dans un champ uniforme



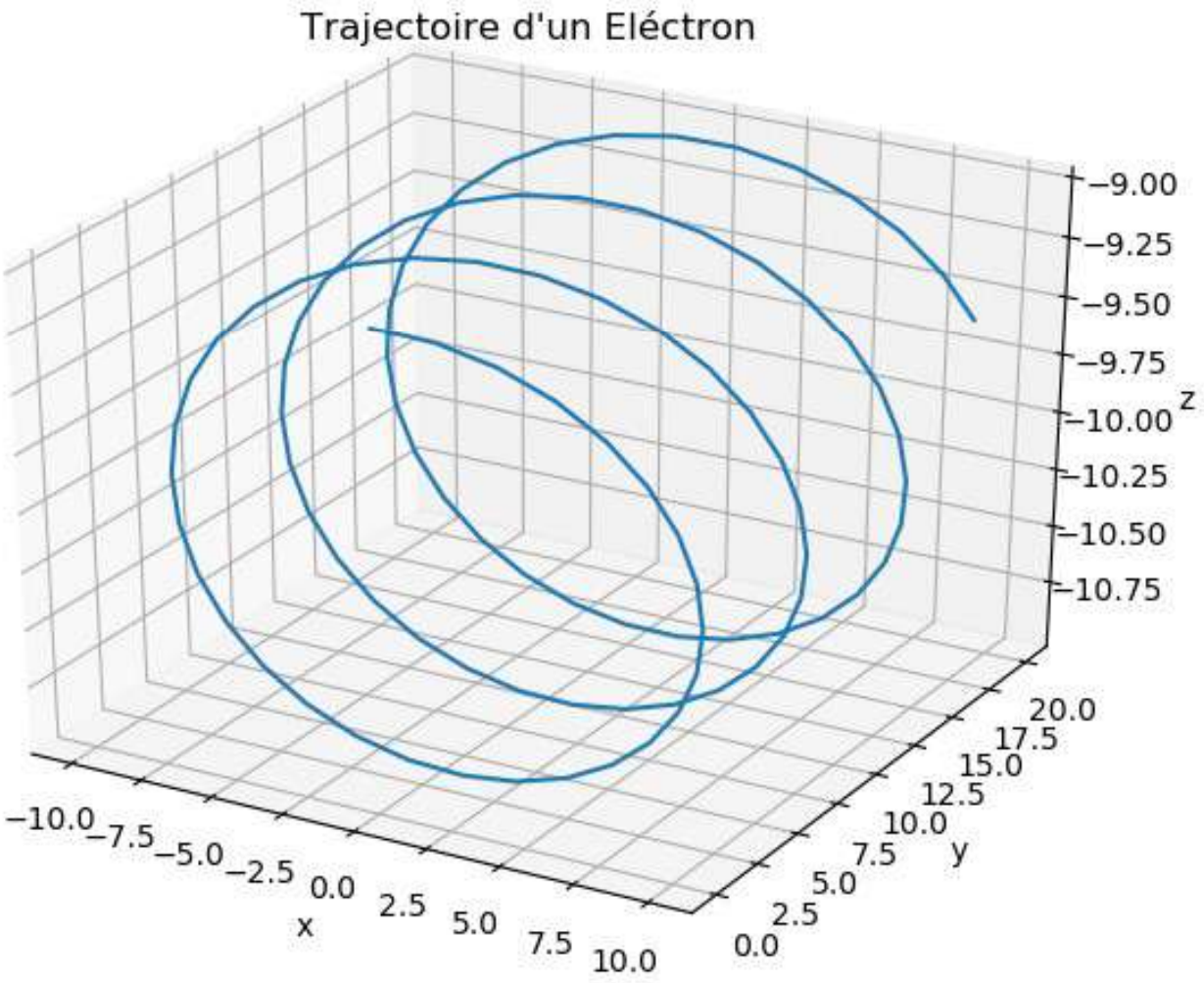
Système: Particule chargée de charge q et de masse m

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

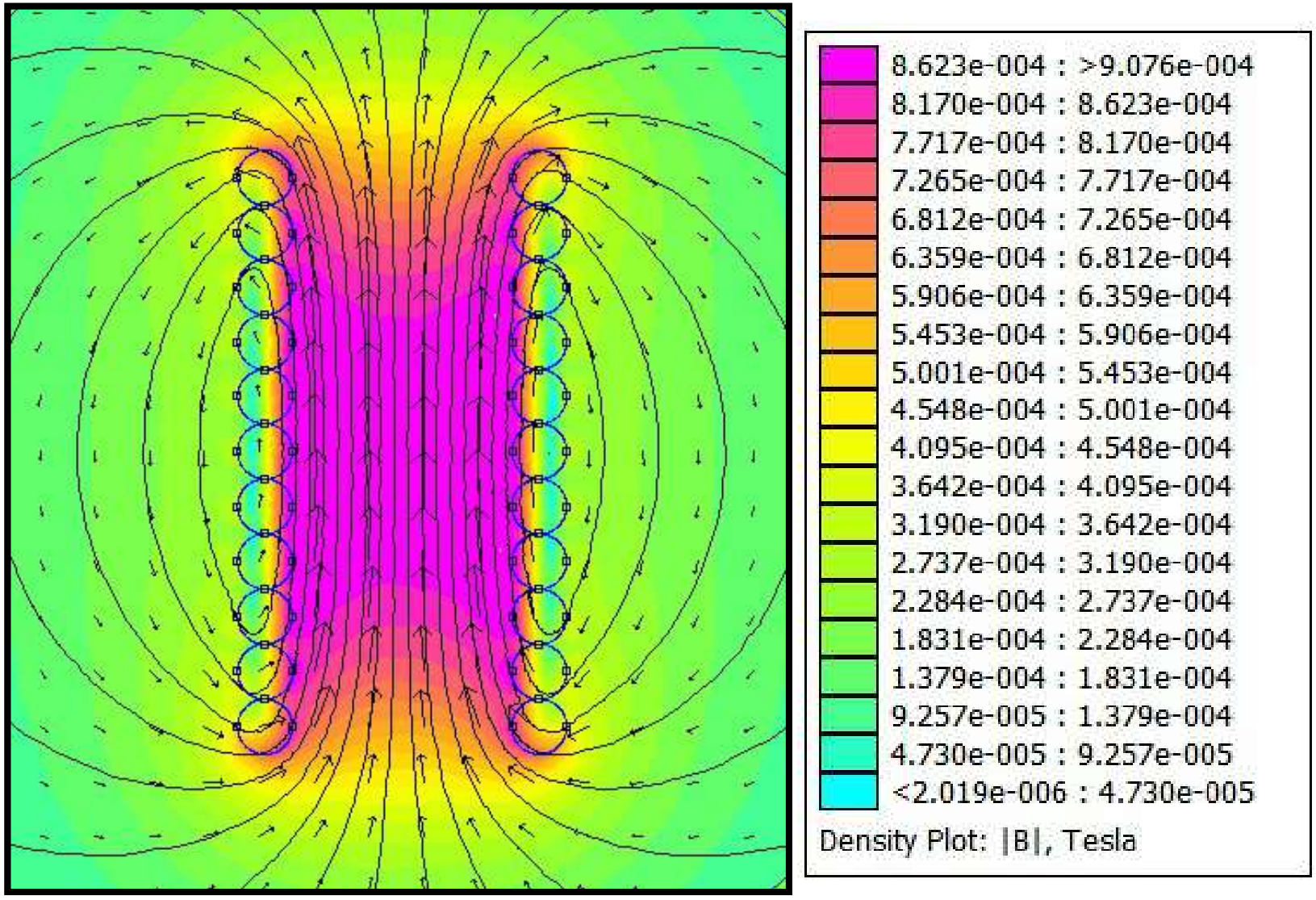
Principe fondamental de la dynamique: $m\dfrac{d\vec{v}}{dt} = q\vec{v} \wedge \vec{B}$

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

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

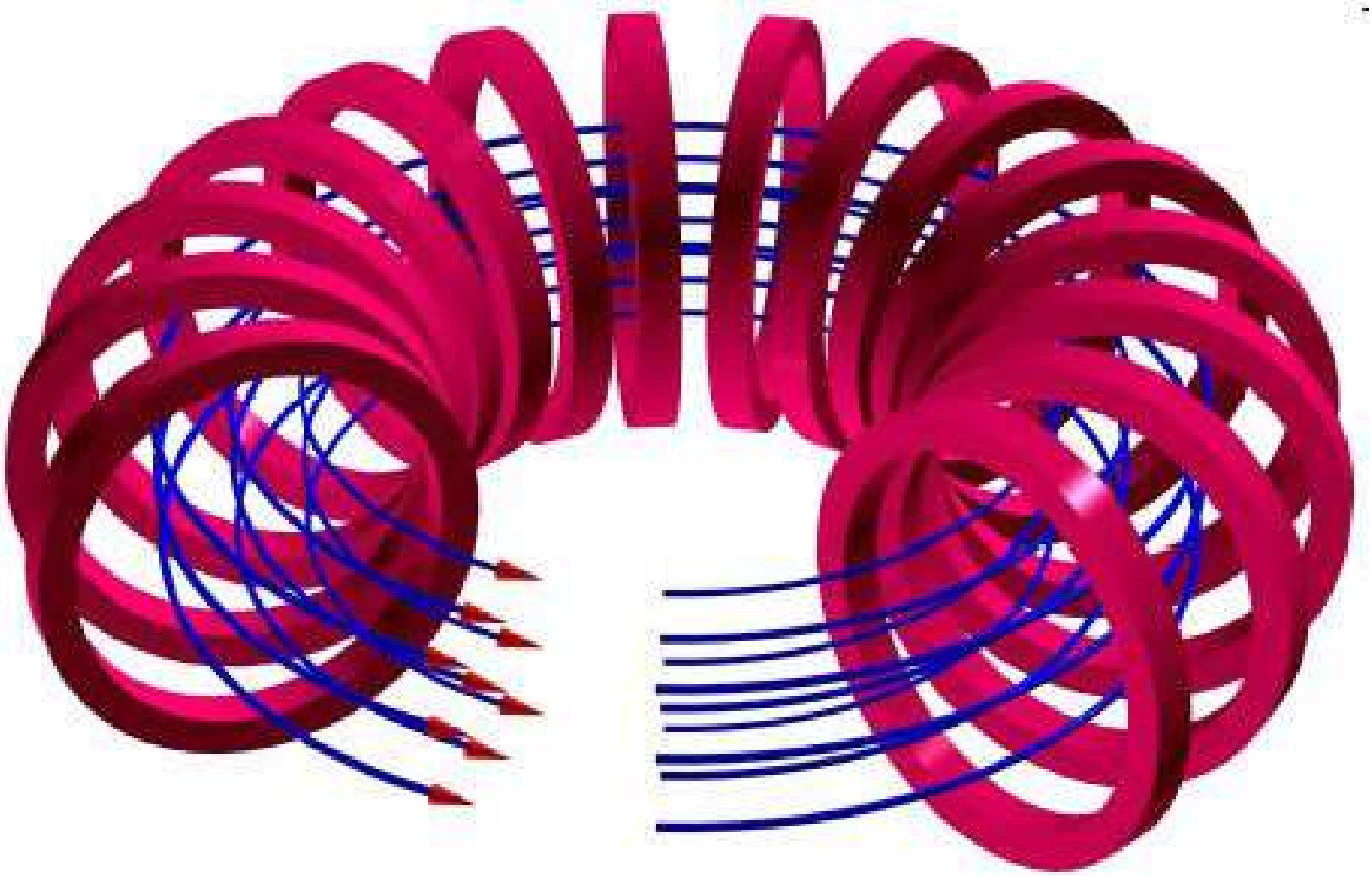


Représentation des lignes de champ d'un solénoïde avec le logiciel FEMM



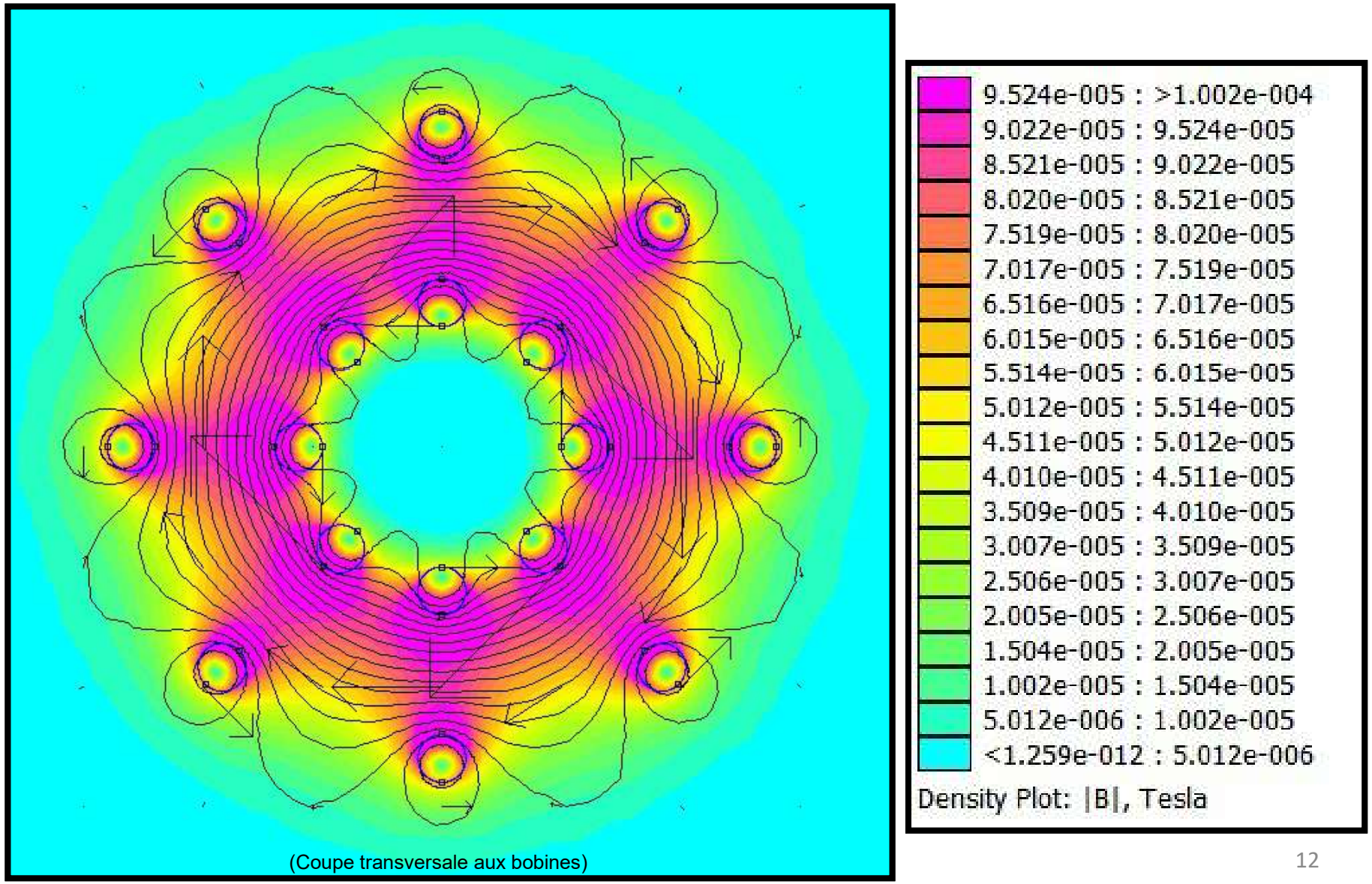
(Coupe transversale aux spires)

Schéma des lignes de champ dans un solénoïde torique



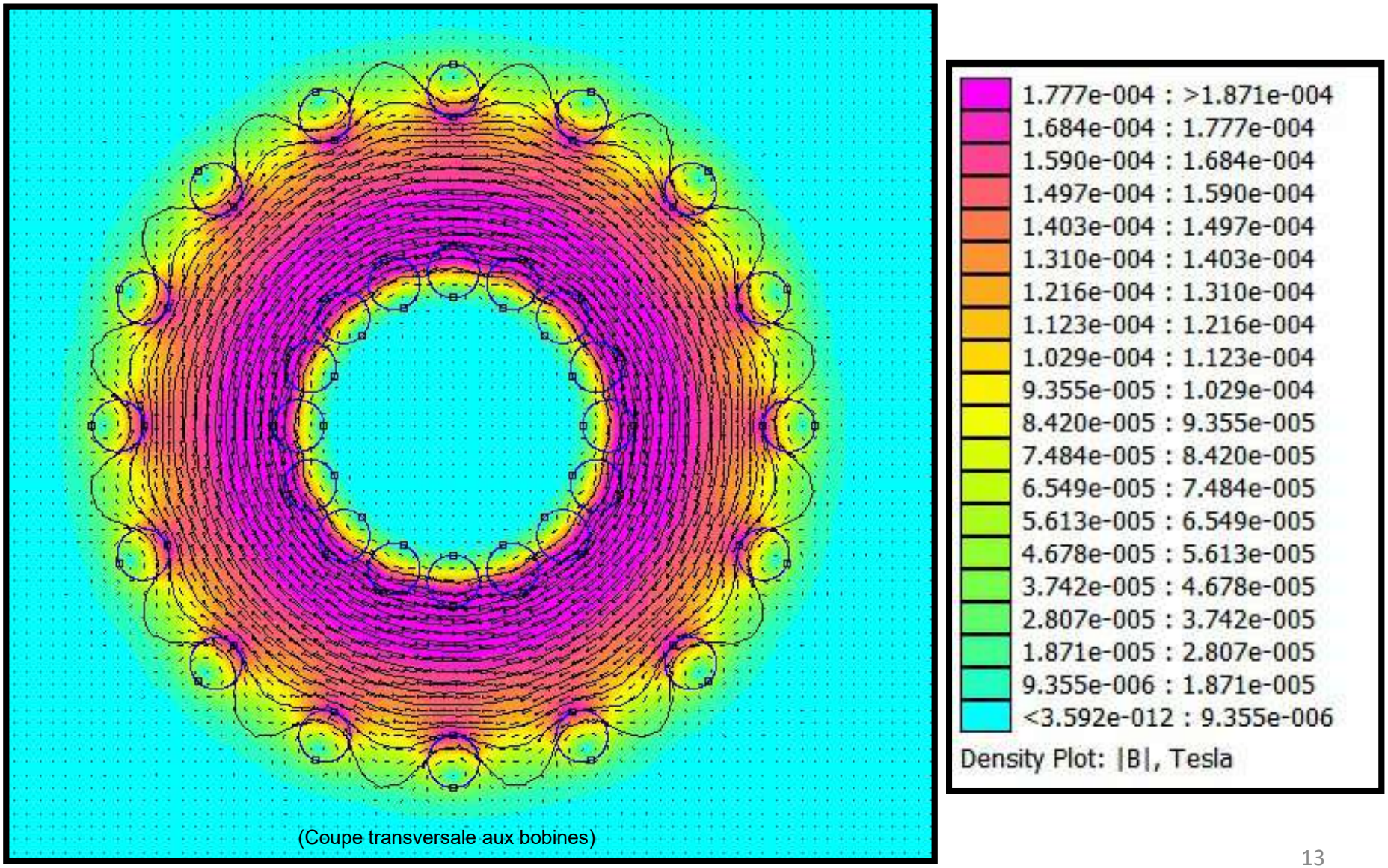
Source: www.fusion-magnétique.cea.fr

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



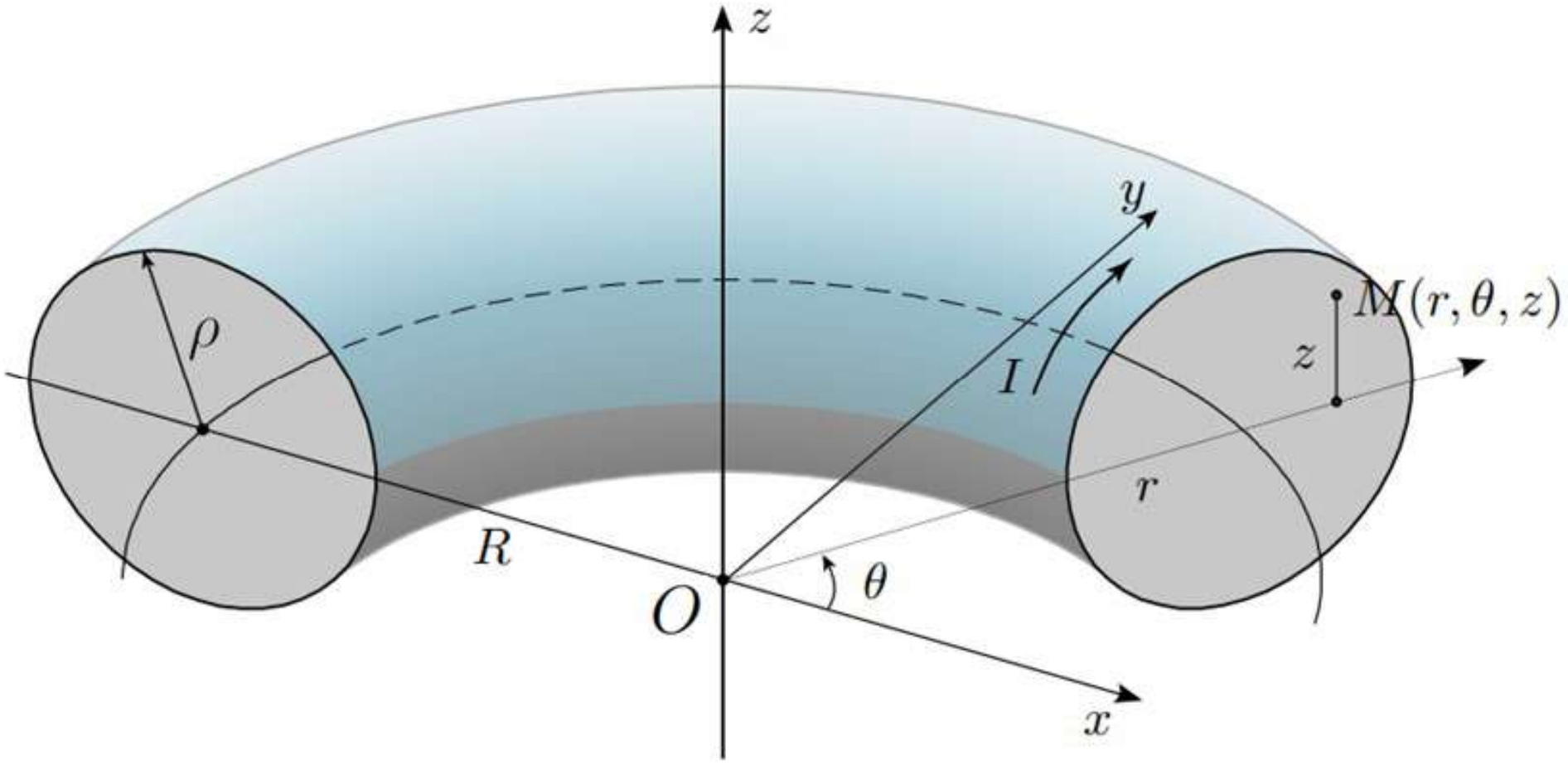
(Coupe transversale aux bobines)

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



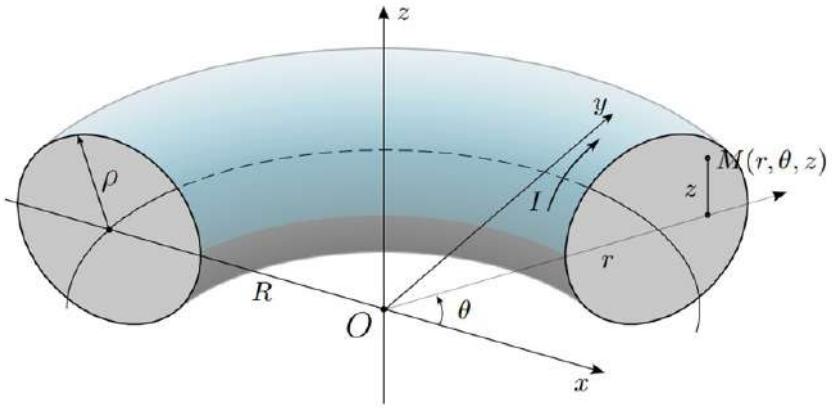
(Coupe transversale aux bobines)

II: Etude d'un Tokamak

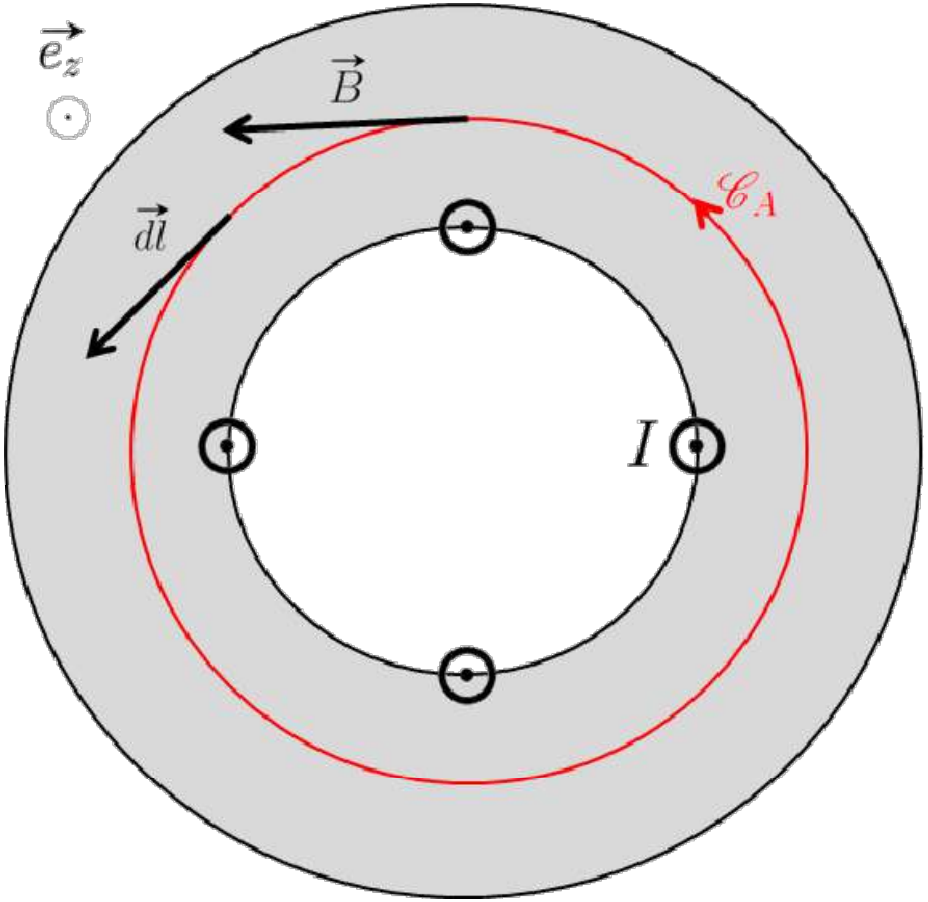


Représentation du tokamak assimilé à un tore et du système de coordonnées utilisé

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: $m\dfrac{d\vec{v}}{dt} = q\vec{v} \wedge \vec{B}$

$$(*) \begin{cases} m\left(\ddot{r} - r\dot{\theta}^2\right) = -\dfrac{q\mu_0 \mathrm{NI}}{2\pi r}\dot{z} \\ 2\dot{r}\dot{\theta} + r\ddot{\theta} = 0 \\ m\ddot{z} = \dfrac{q\mu_0 \mathrm{NI}}{2\pi r}\dot{r} \end{cases}$$

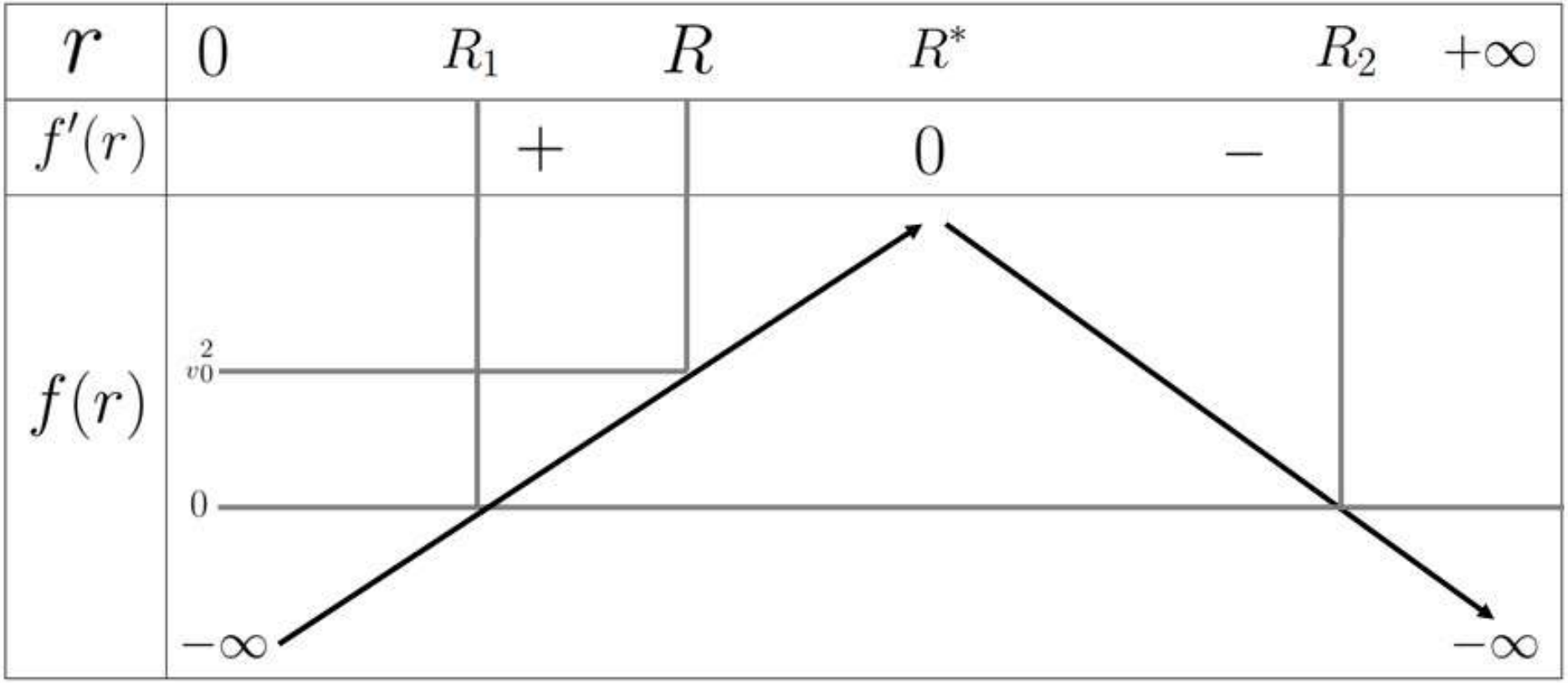
Conditions initiales: $\begin{cases} r_0 = R, \theta_0 = 0, z_0 = 0 \\ \dot{r_0} = v_0, \dot{\theta_0} = \dfrac{v_0}{R}, \dot{z_0} = 0 \end{cases}$

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

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

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

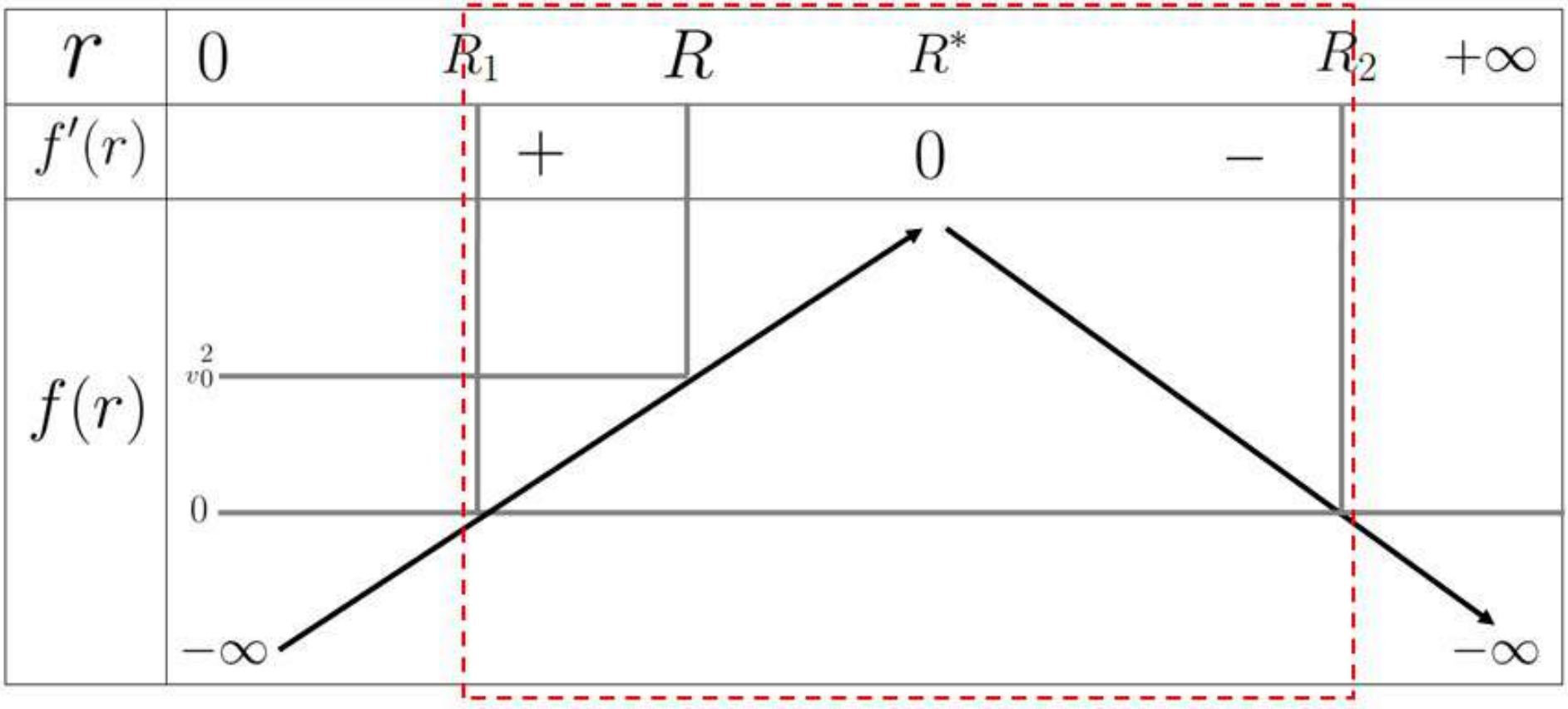


Tableau regroupant les caractéristiques de deux tokamaks

	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 »			

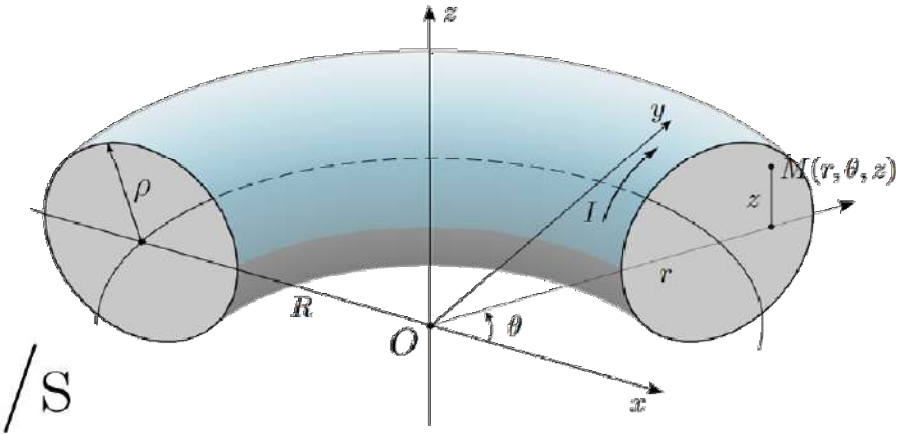
Choix des paramètres

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: $\begin{cases} r_0 = R, \theta_0 = 0, z_0 = 0 \\ \dot{r_0} = v_0, \dot{\theta_0} = \dfrac{v_0}{R}, \dot{z_0} = 0 \end{cases}$

$(*)\ devient :$ $\boxed{\begin{cases} \ddot{r} = \dfrac{(Rv_0)^2}{r^3} - \dfrac{\beta^2}{r} \ln \dfrac{r}{R} \\ \dot{\theta} = \dfrac{Rv_0}{r^2} \\ \dot{z} = \beta \ln \dfrac{r}{R} \end{cases}}$

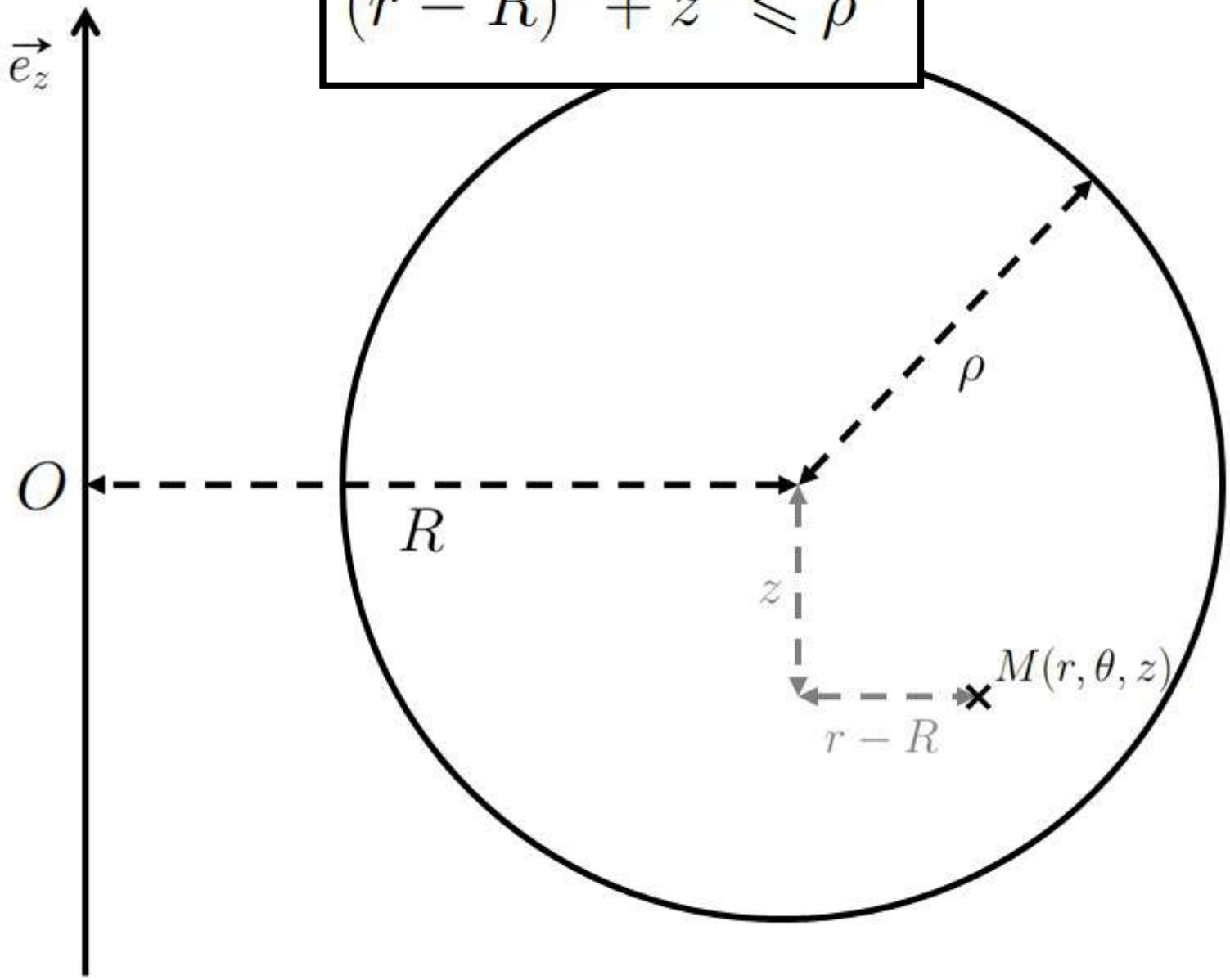
Mise en place d'une procédure numérique pour résoudre par approximation, des équations différentielles du premier ordre avec une condition initiale

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

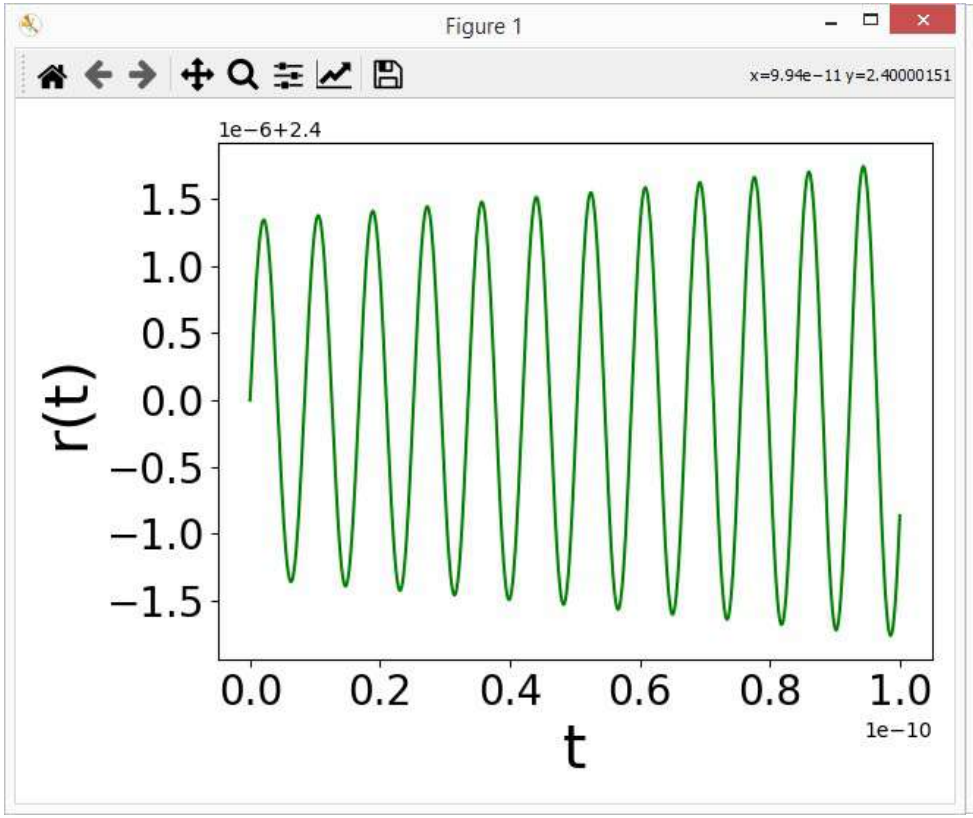
$$\begin{cases} r_{n+1} = r_n + \tau\, s_n \\ s_{n+1} = s_n + \tau \left(\frac{(Rv_0)^2}{r^3} - \frac{\beta^2}{r} \ln \frac{r}{R} \right) \\ \theta_{n+1} = \theta_n + \tau \frac{Rv_0}{r^2} \\ z_{n+1} = z_n + \tau\, \beta \ln \frac{r}{R} \end{cases}$$

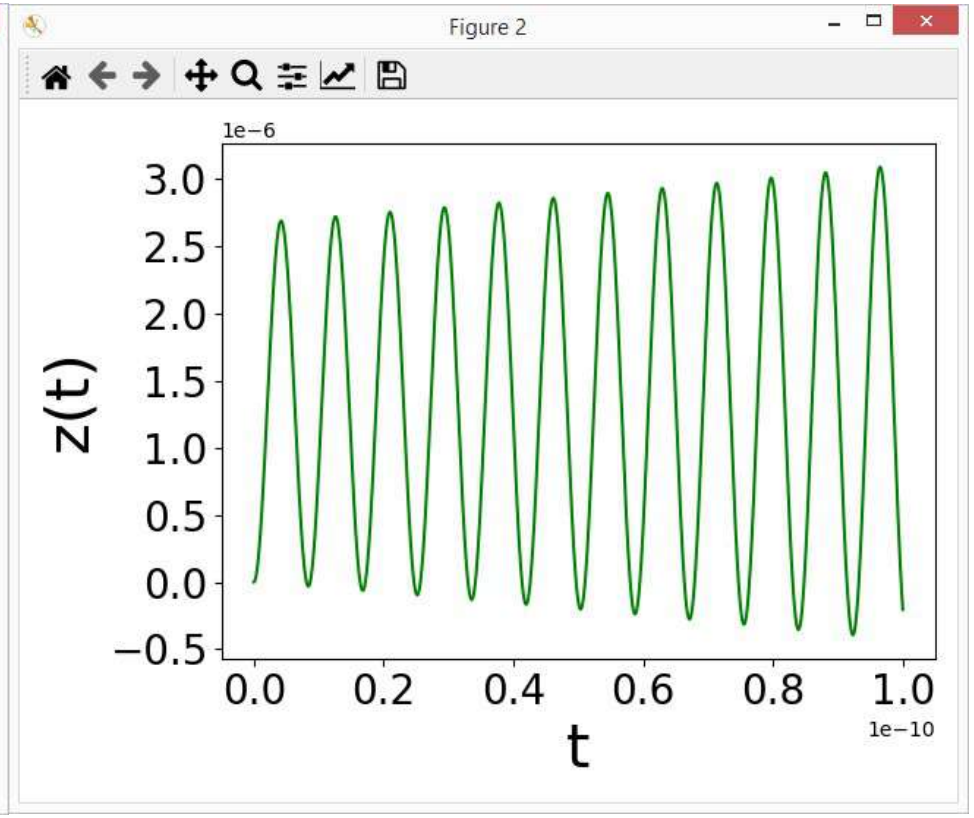
Condition pour rester dans le tokamak

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

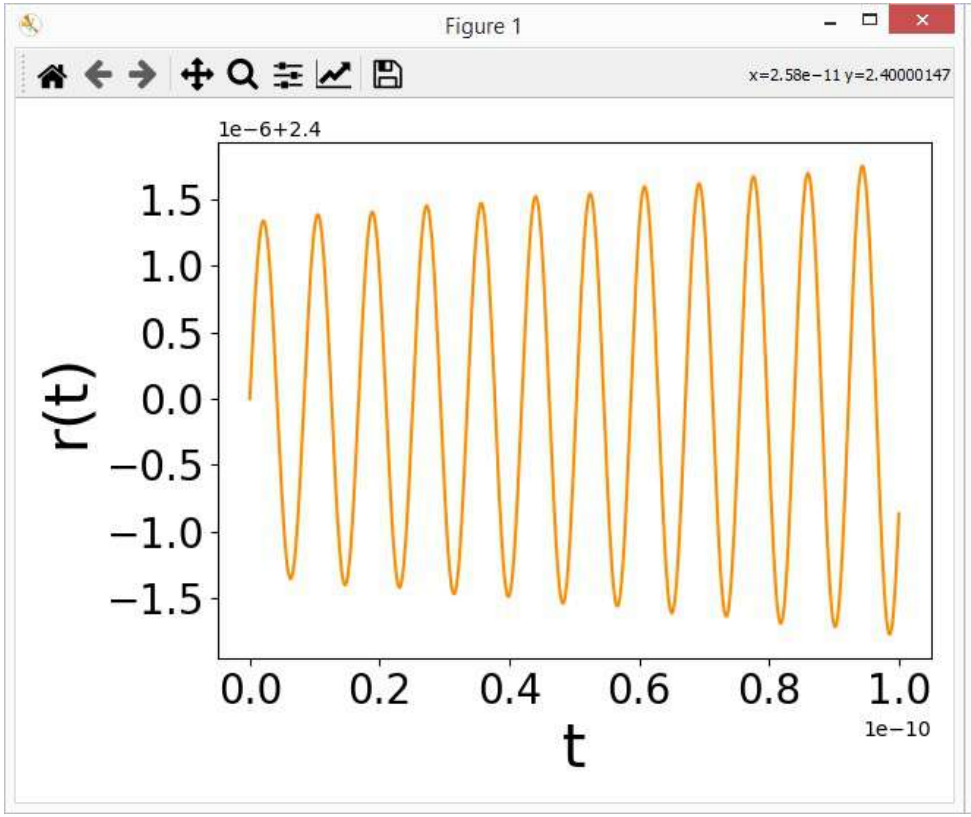


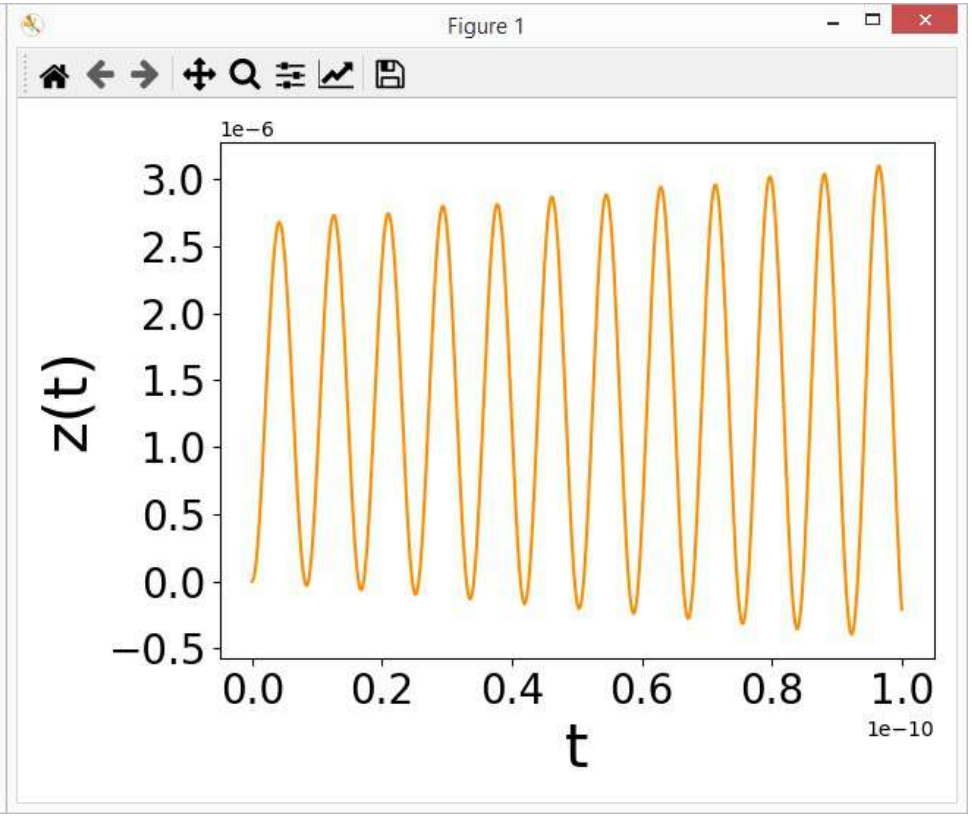
Résultats Obtenus avec la Méthode d'Euler



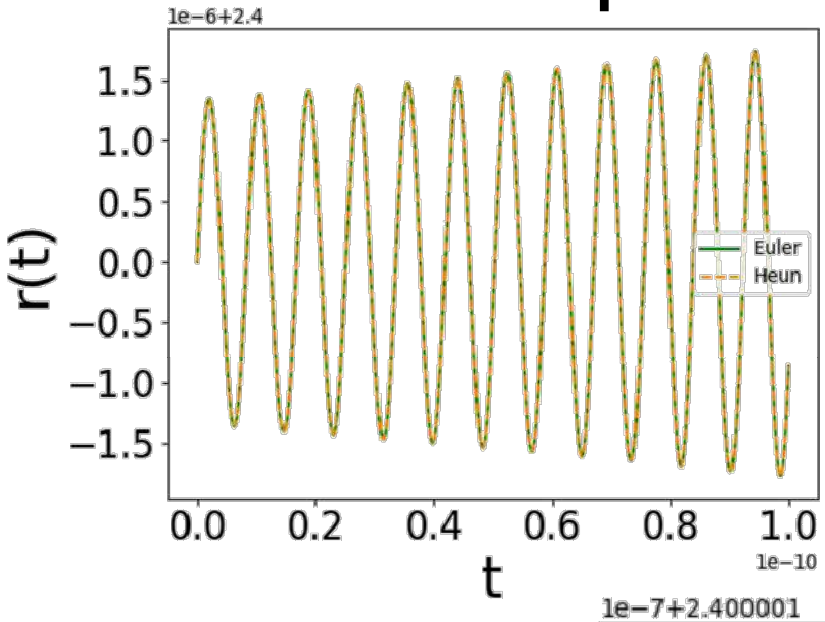


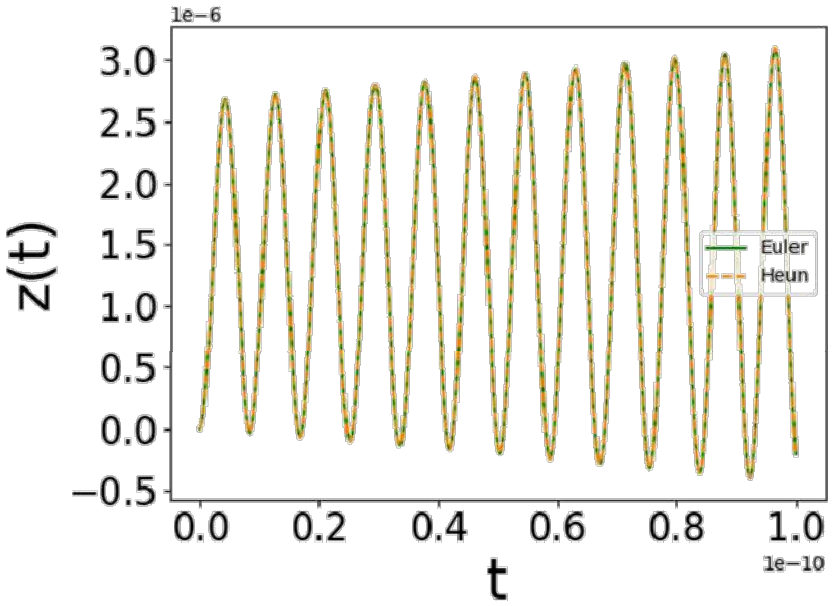
Résultats Obtenus avec la Méthode de Heun

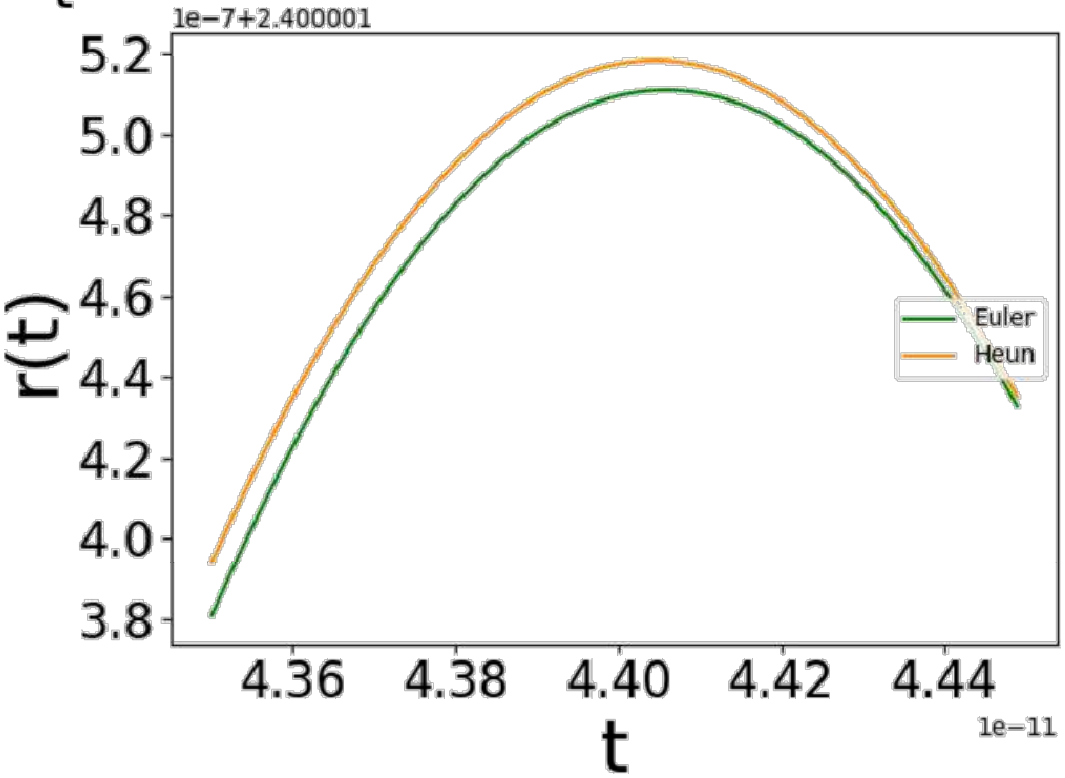




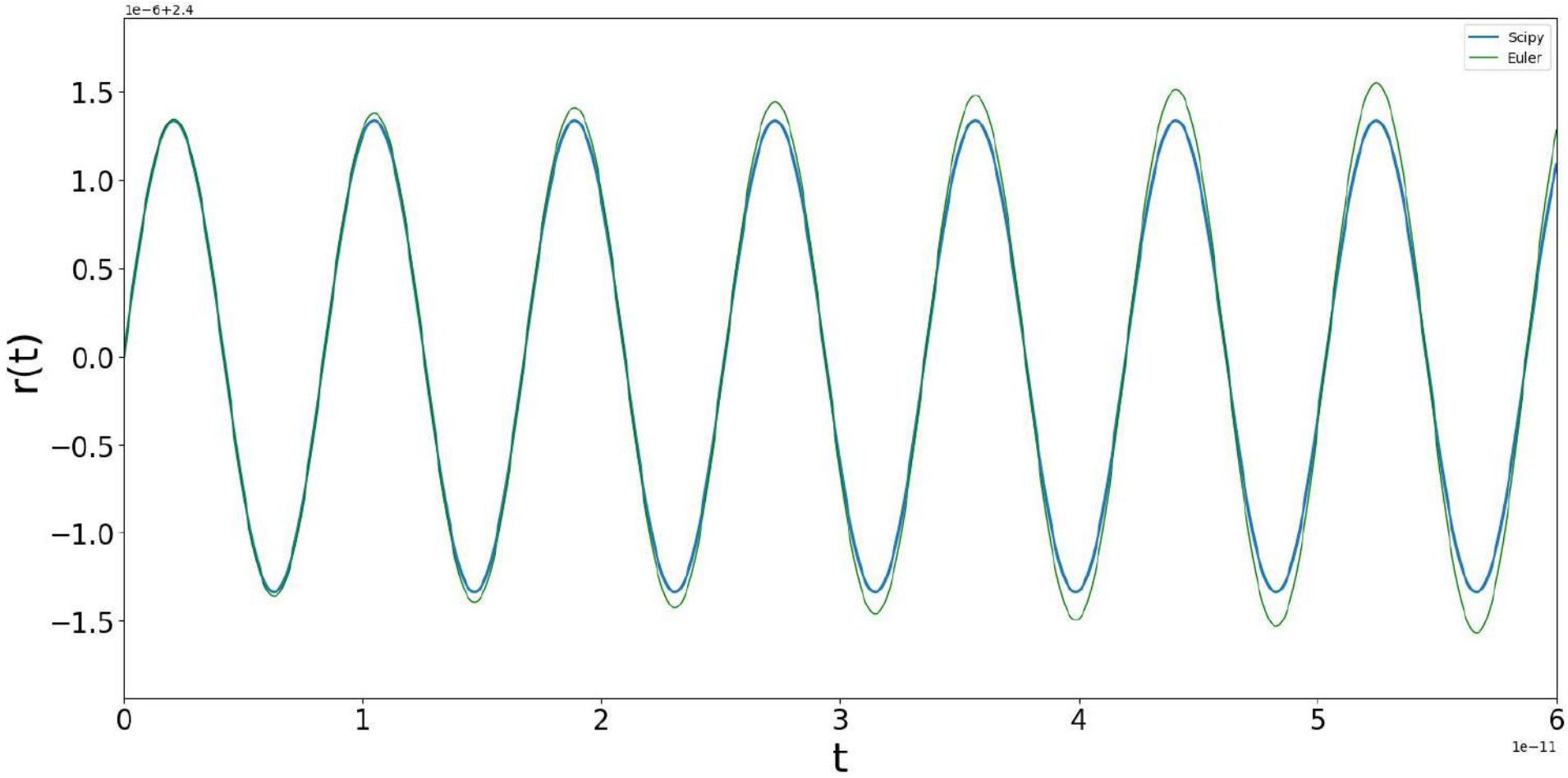
Comparaison des deux Méthodes



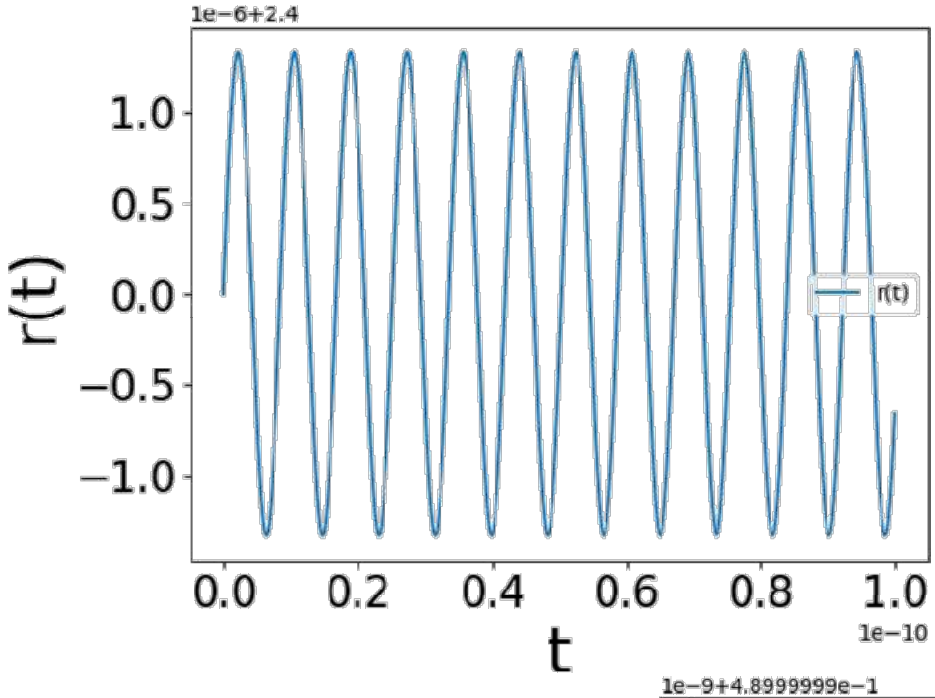


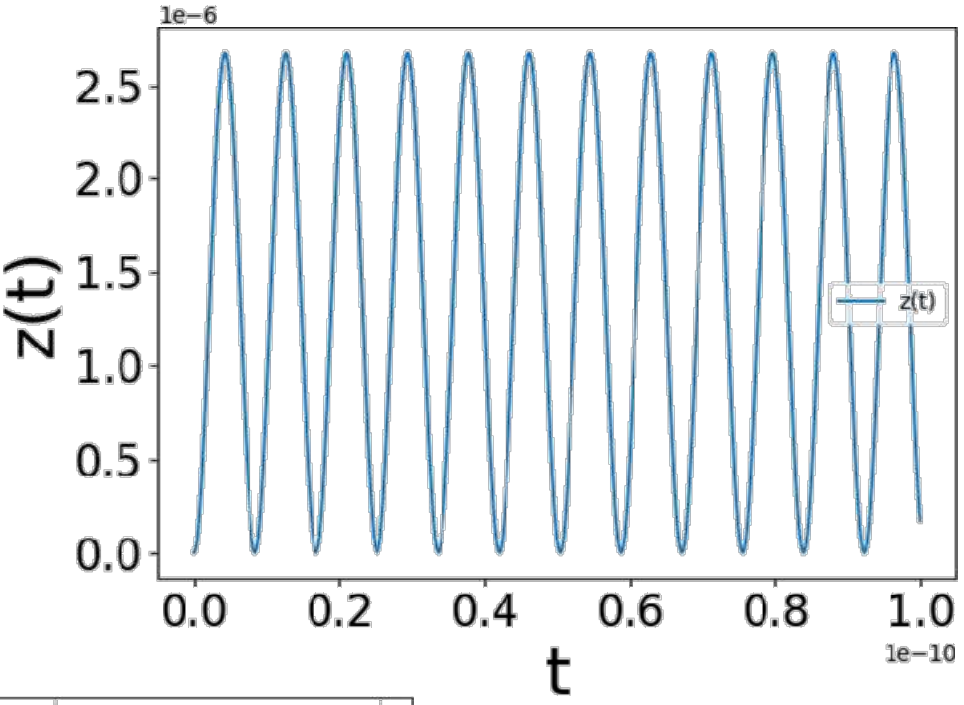


Comparaison de la méthode d'Euler et odeint

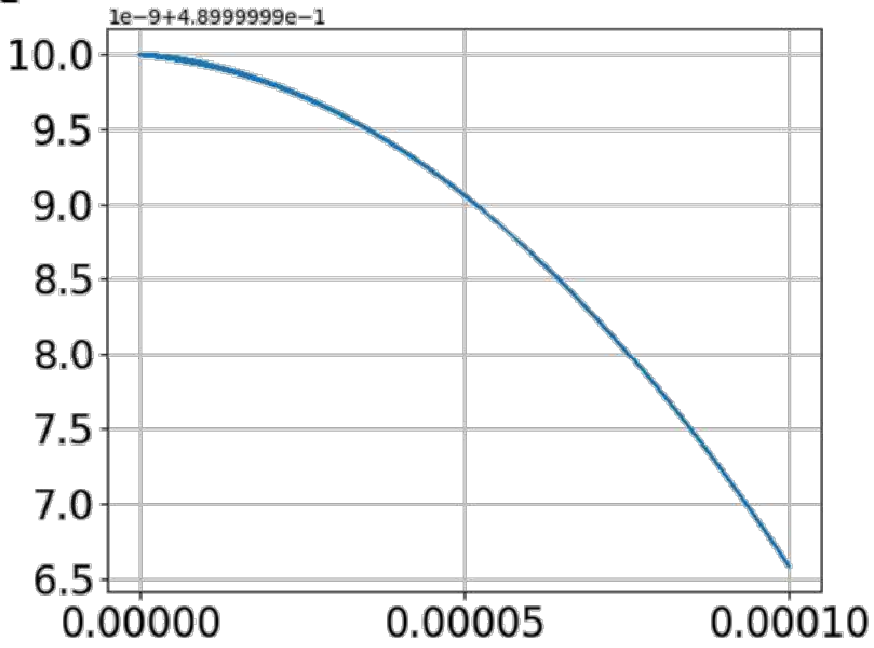


Résultats Obtenus avec le module Scipy

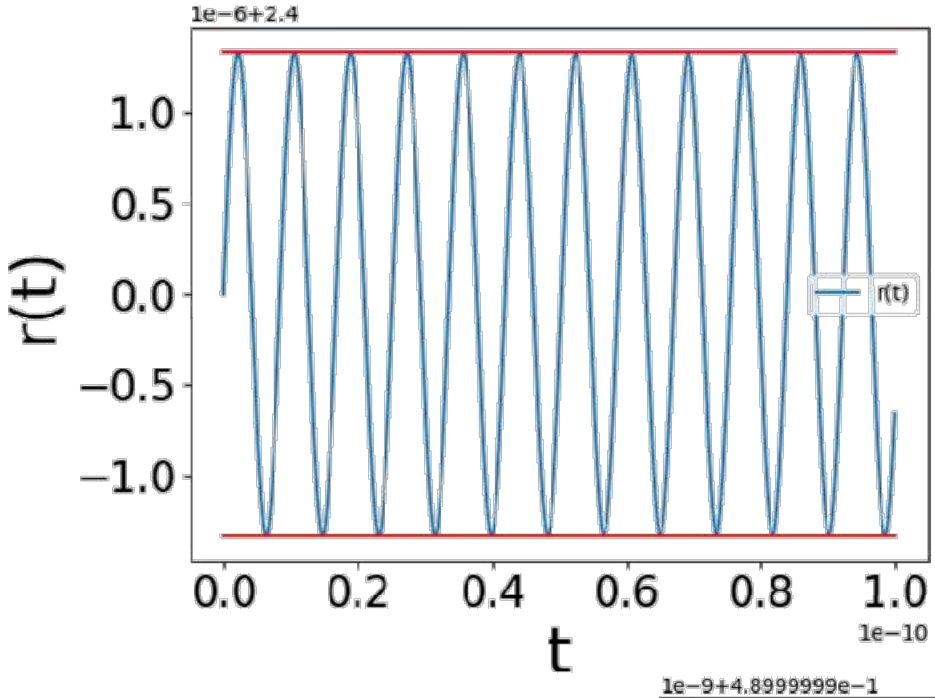


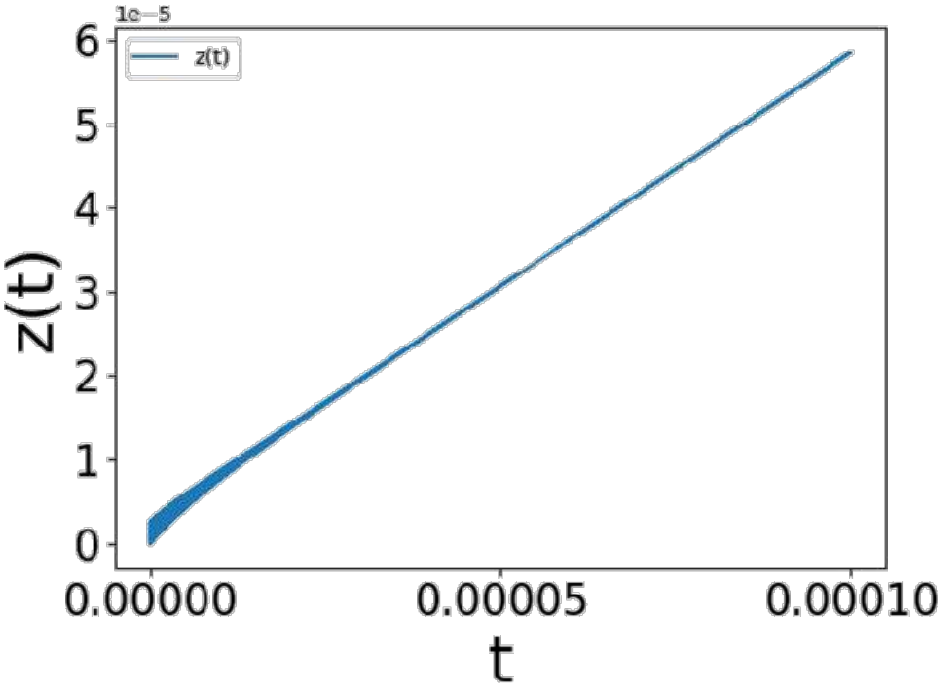


$$0 \leqslant \rho^2 - (r - R)^2 - z^2$$

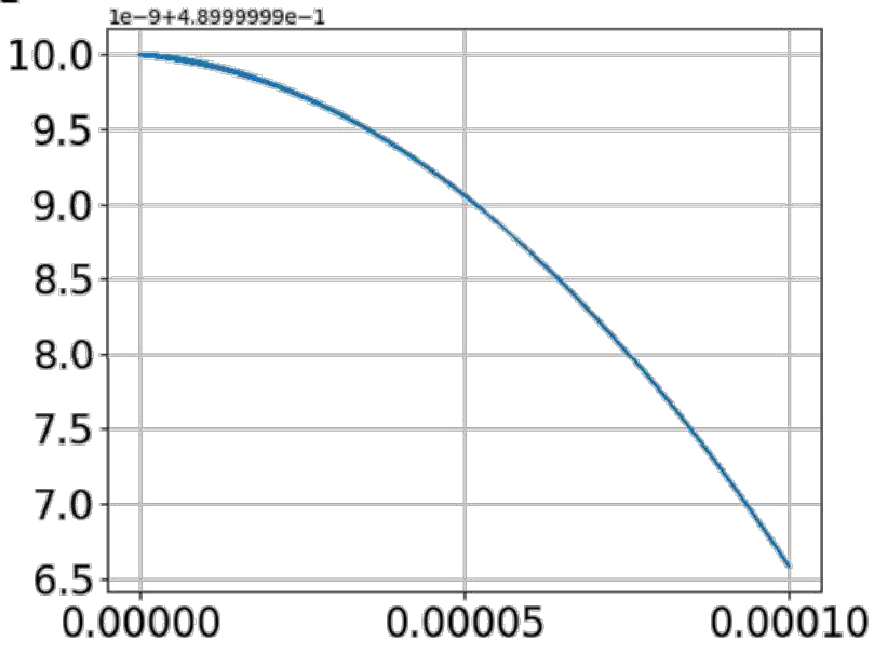


Résultats Obtenus avec le module Scipy

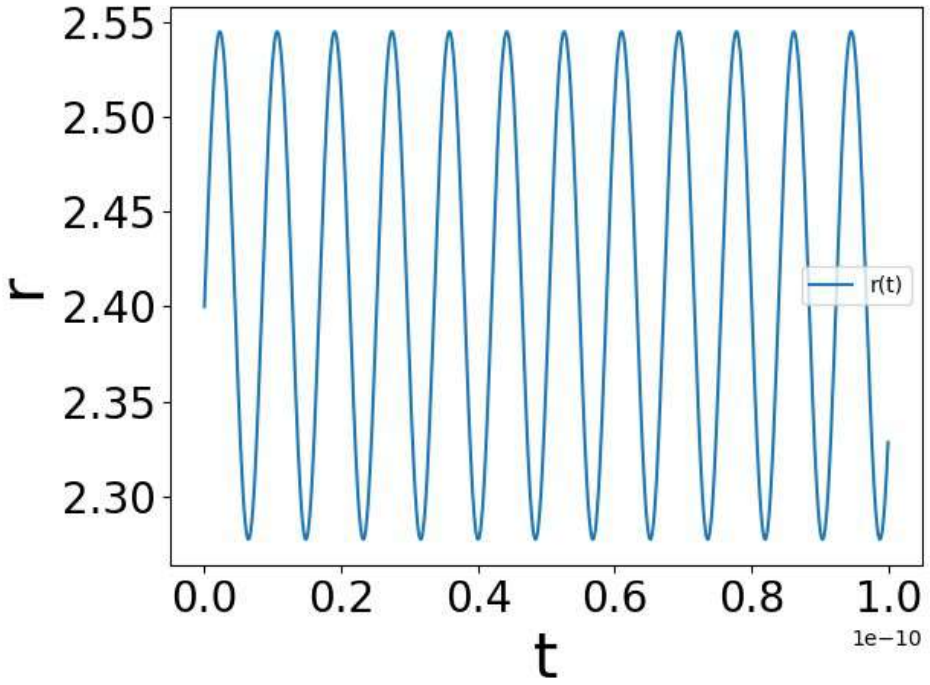


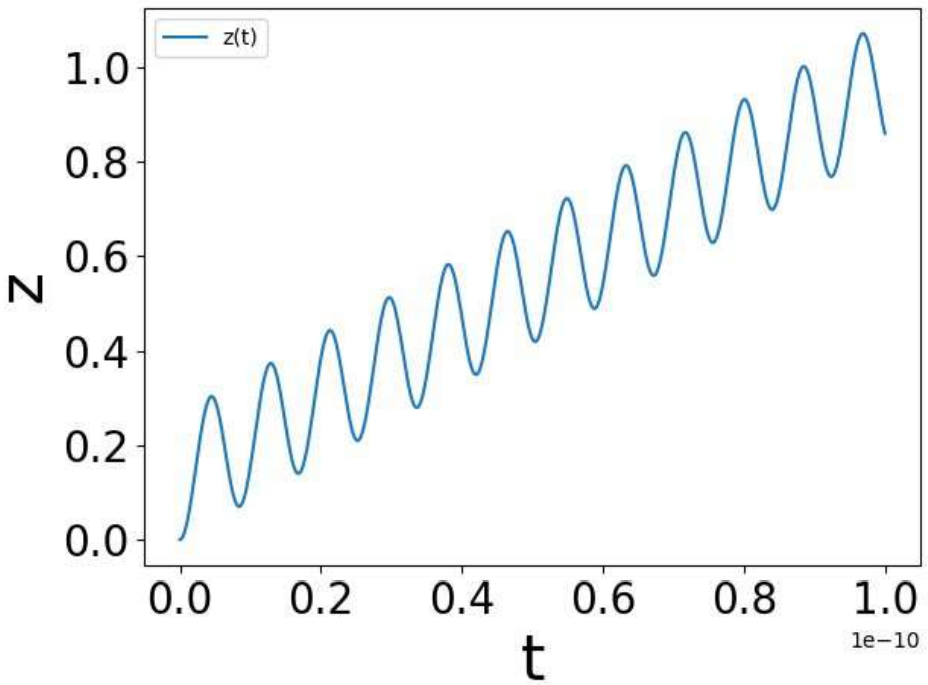


$$0 \leqslant \rho^2 - (r - R)^2 - z^2$$

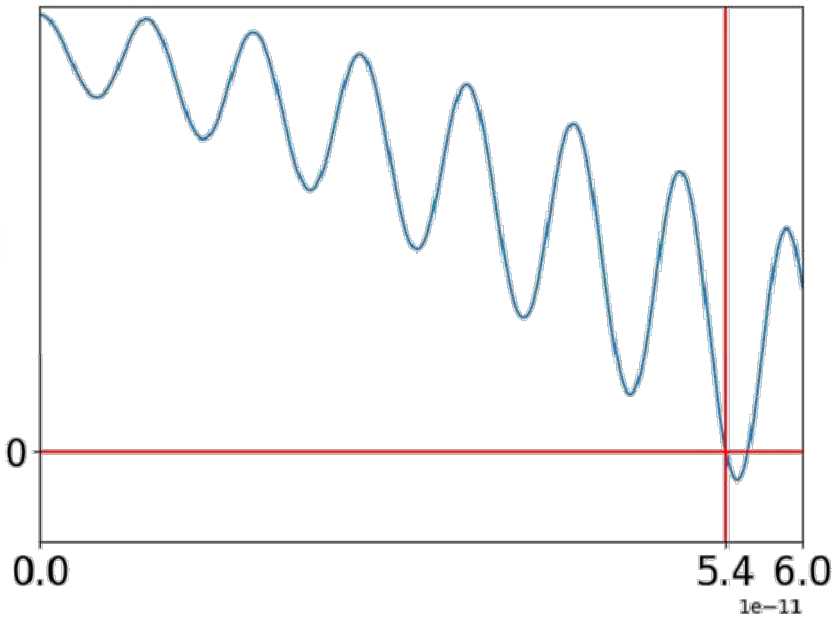


Résultats Obtenus avec le module Scipy



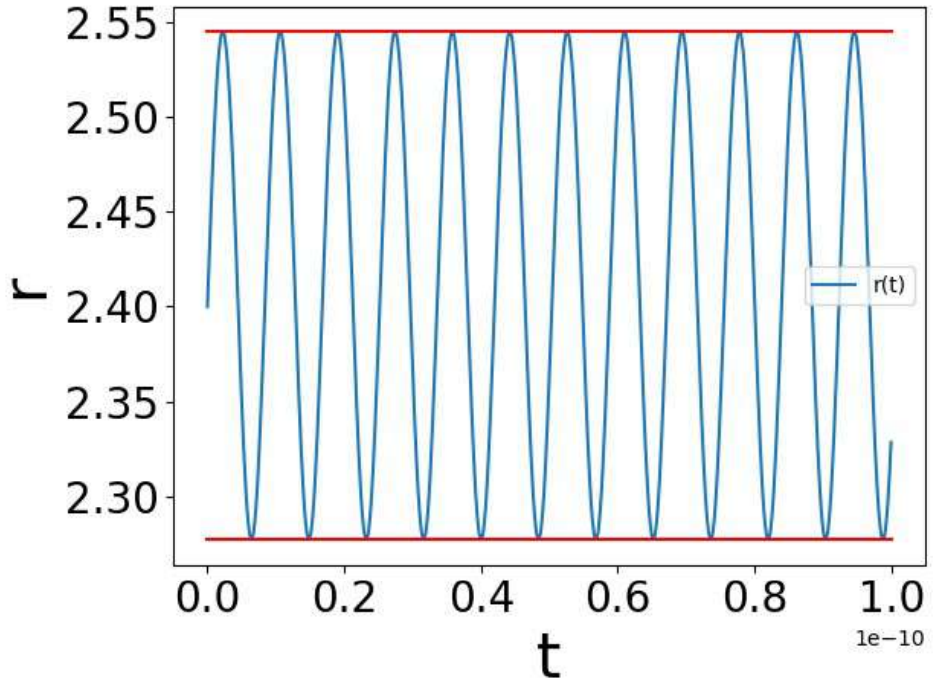


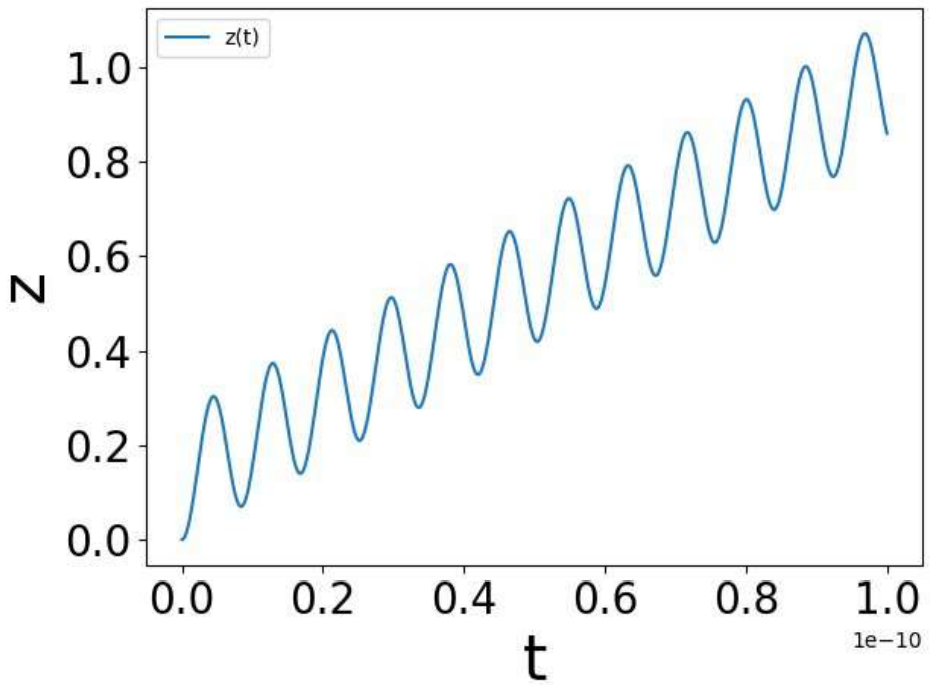
$$0 \leqslant \rho^2 - (r - R)^2 - z^2$$



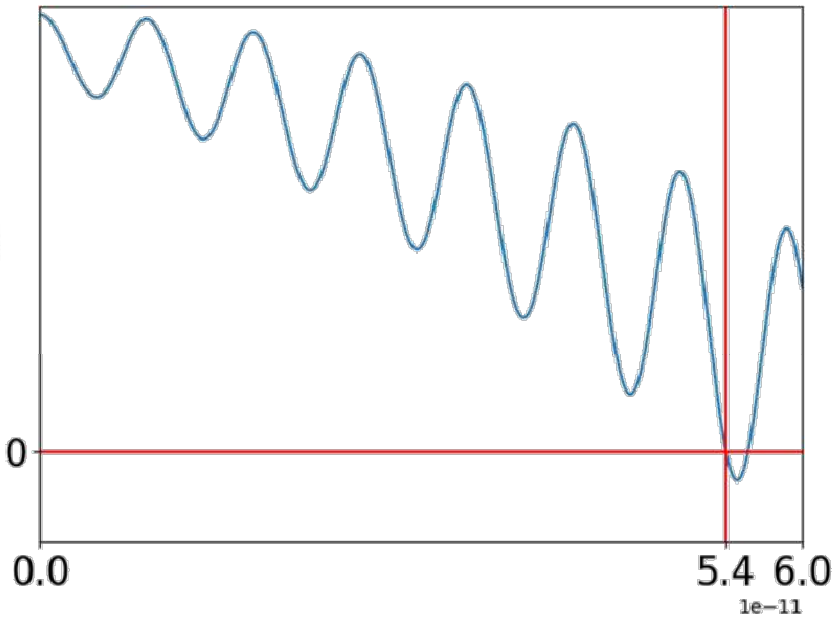
Pour une vitesse initiale plus élevée

Résultats Obtenus avec le module Scipy

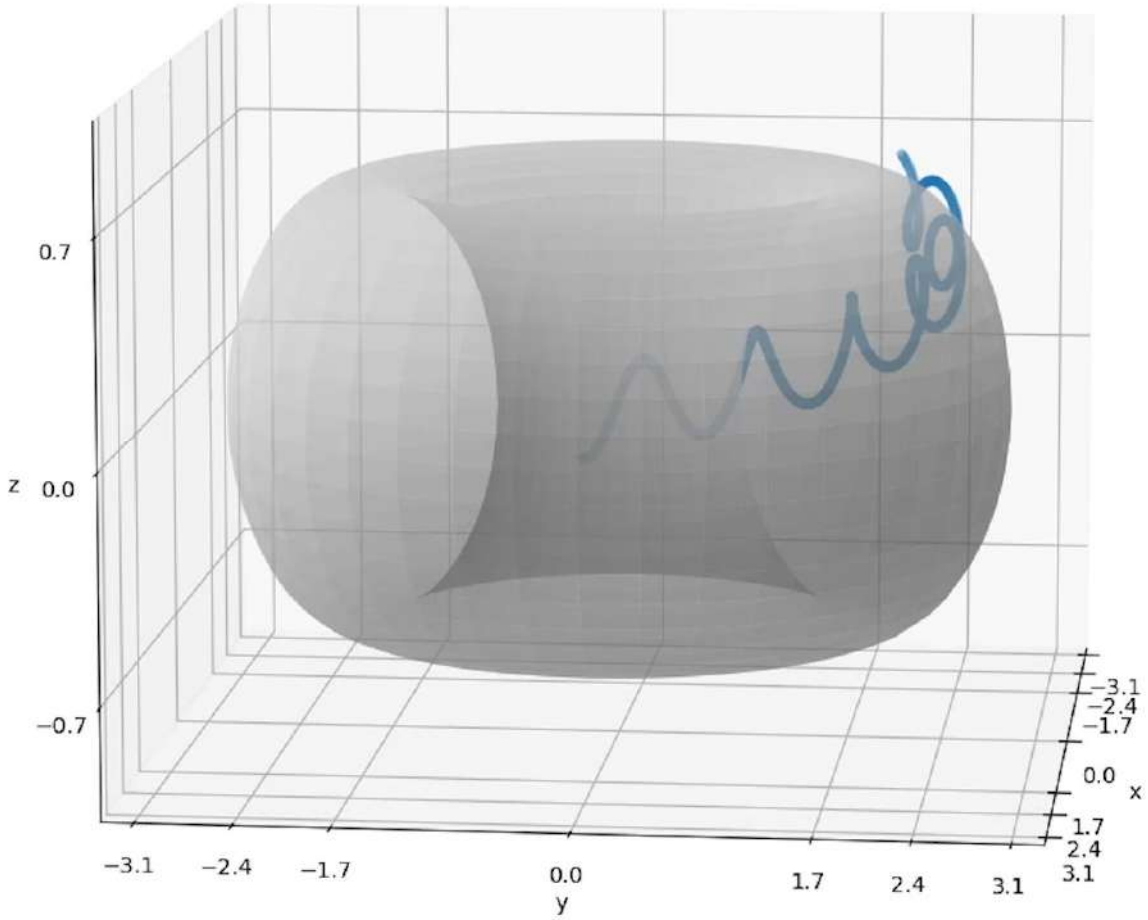


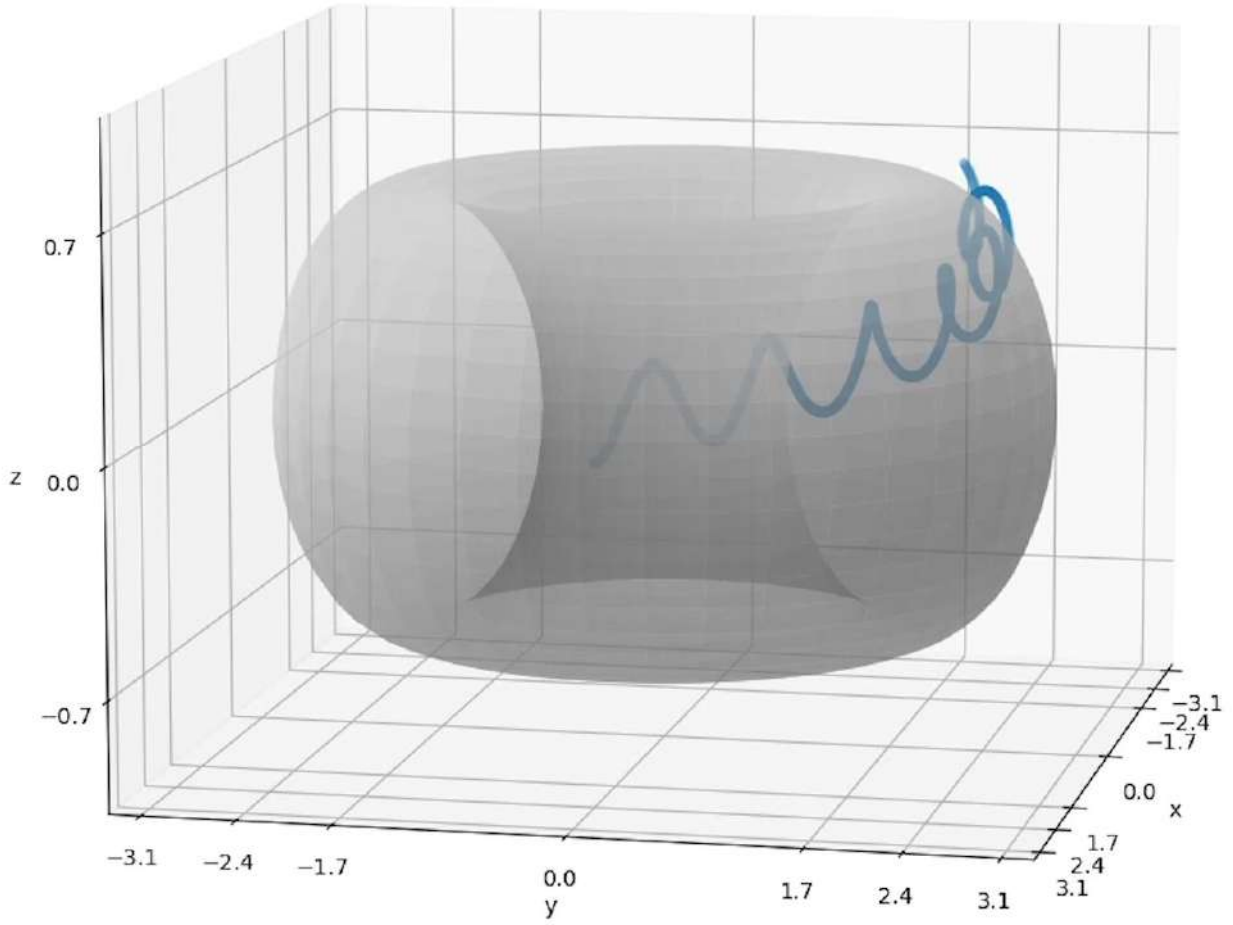


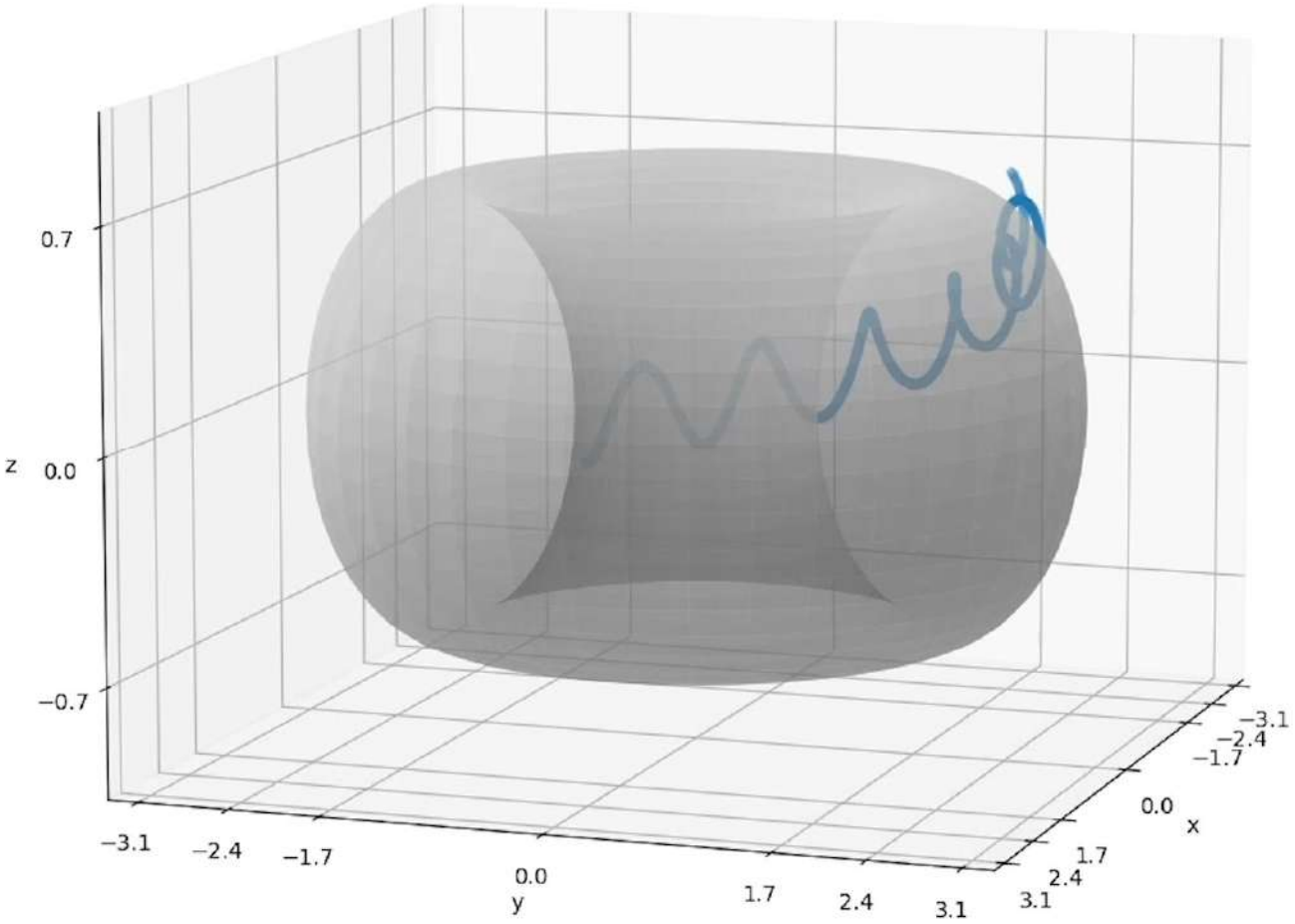
$$0 \leqslant \rho^2 - (r - R)^2 - z^2$$

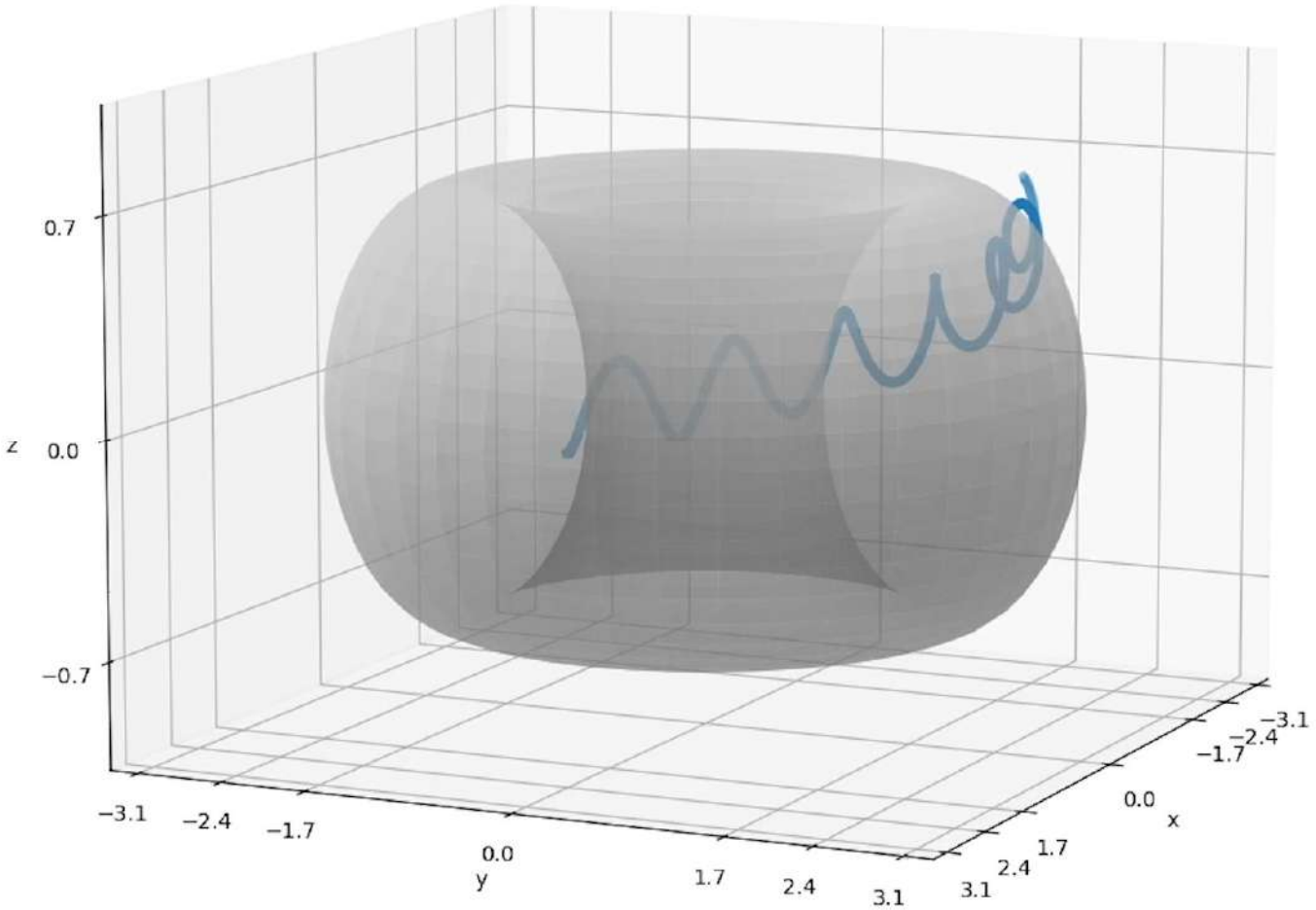


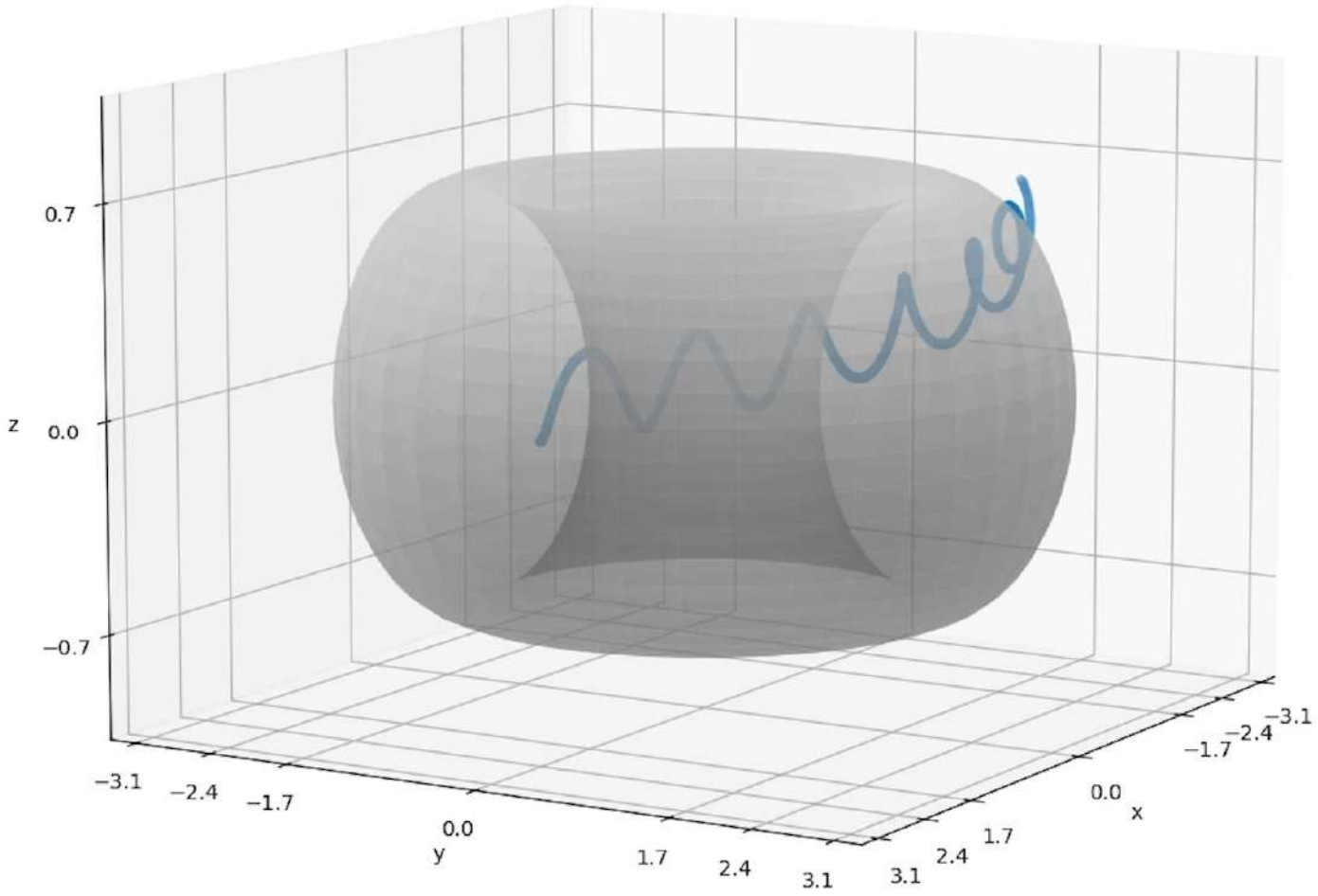
Pour une vitesse initiale plus élevée

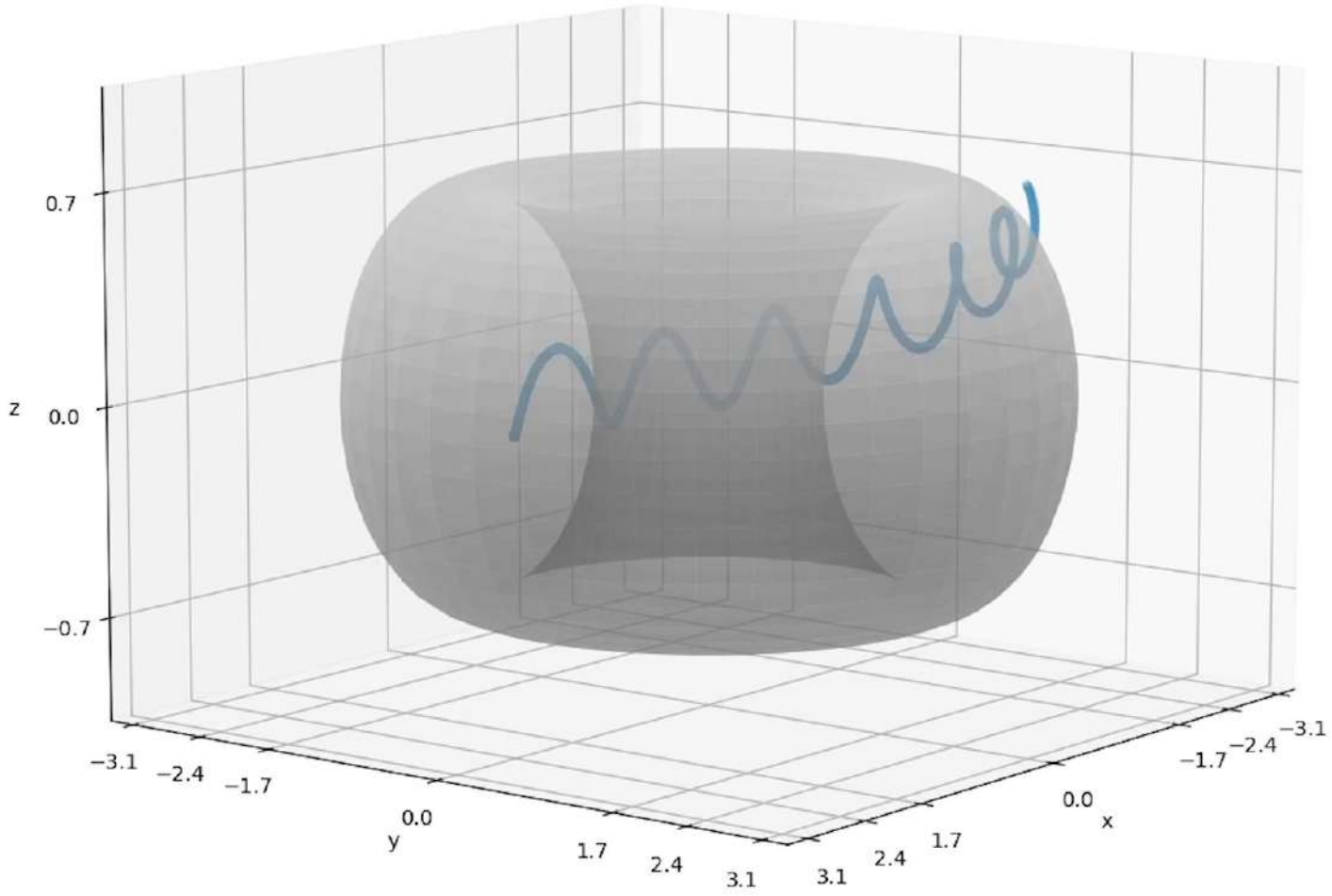

0.7
z 0.0
−0.7
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
−3.1
−2.4
−1.7
0.0
x
1.7
2.4
3.1

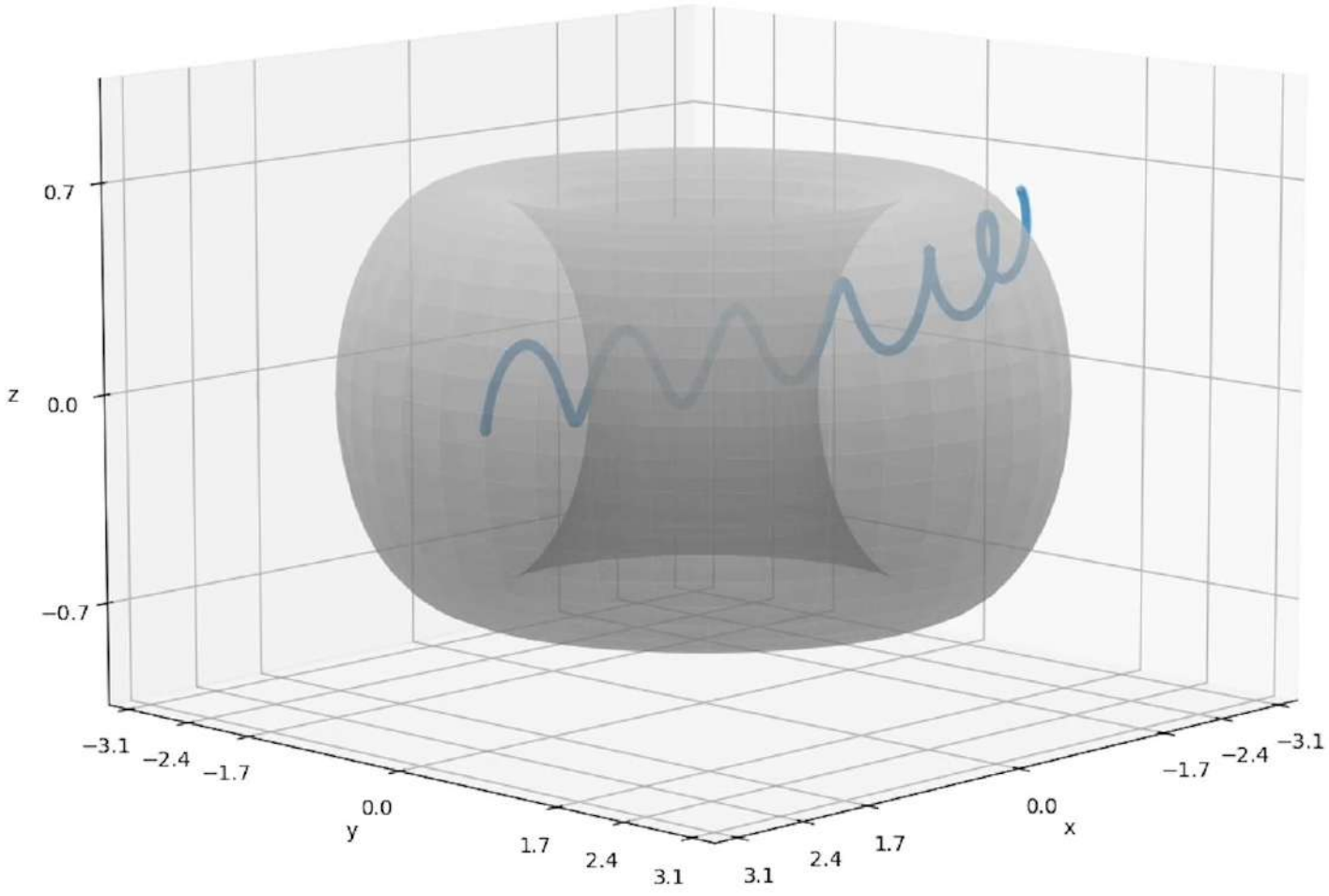

0.7
z
0.0
−0.7
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
−3.1
−2.4
−1.7
0.0
x
1.7
2.4
3.1

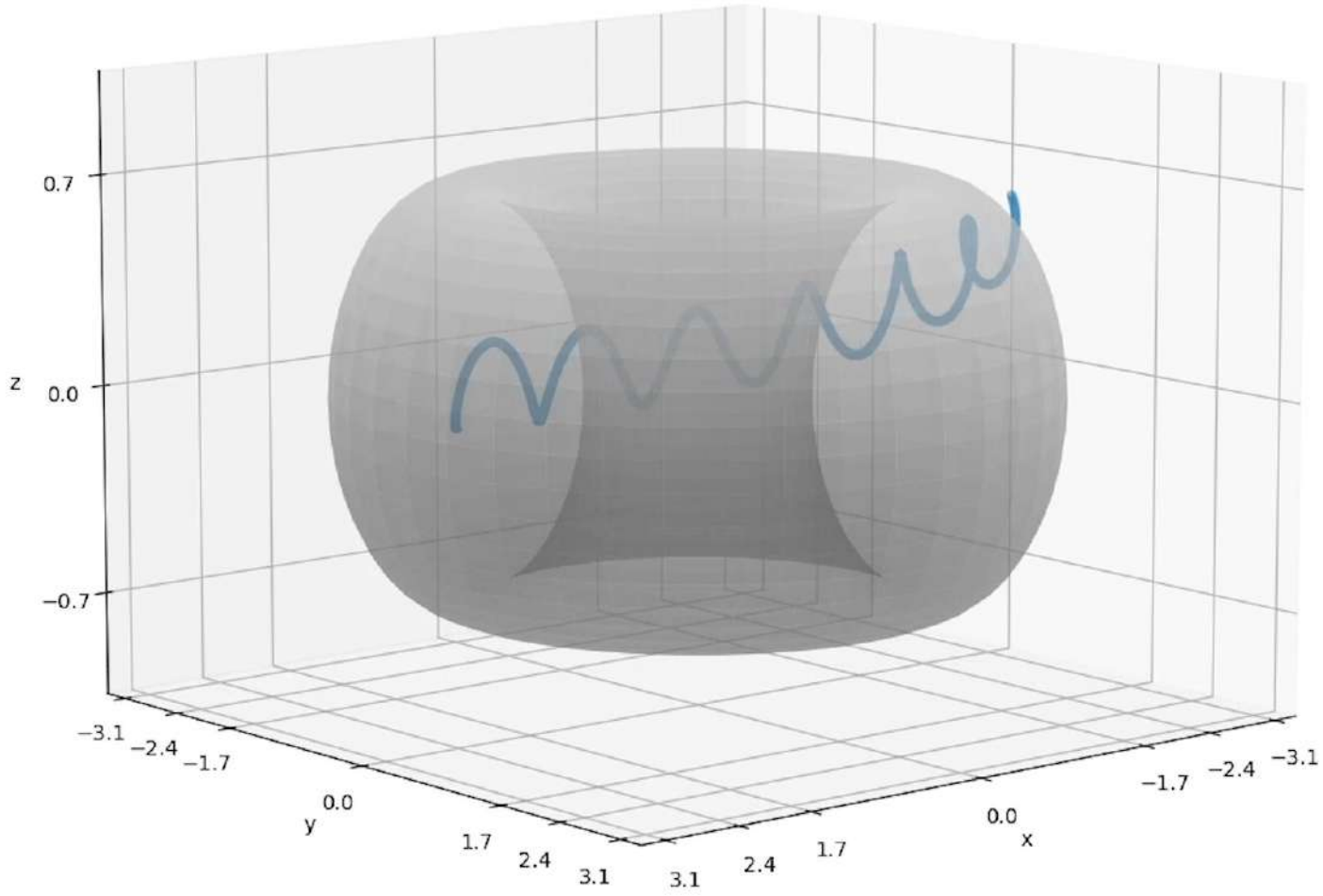

0.7
z
0.0
−0.7
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
−3.1
−2.4
−1.7
0.0
x
1.7
2.4
3.1

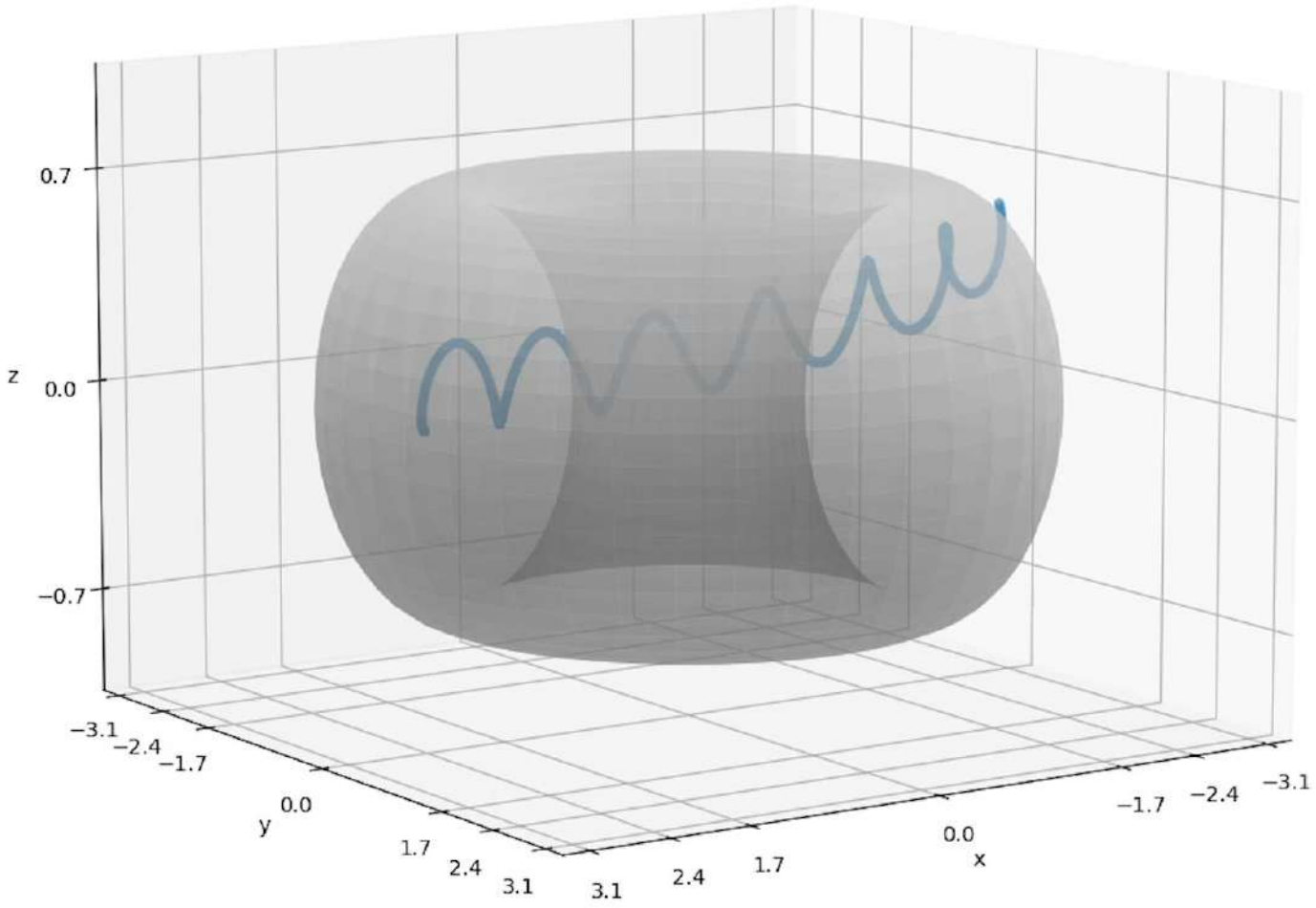

0.7
0.0
−0.7
z
−3.1
−2.4
−1.7
0.0
1.7
2.4
3.1
y
−3.1
−2.4
−1.7
0.0
1.7
2.4
3.1
x

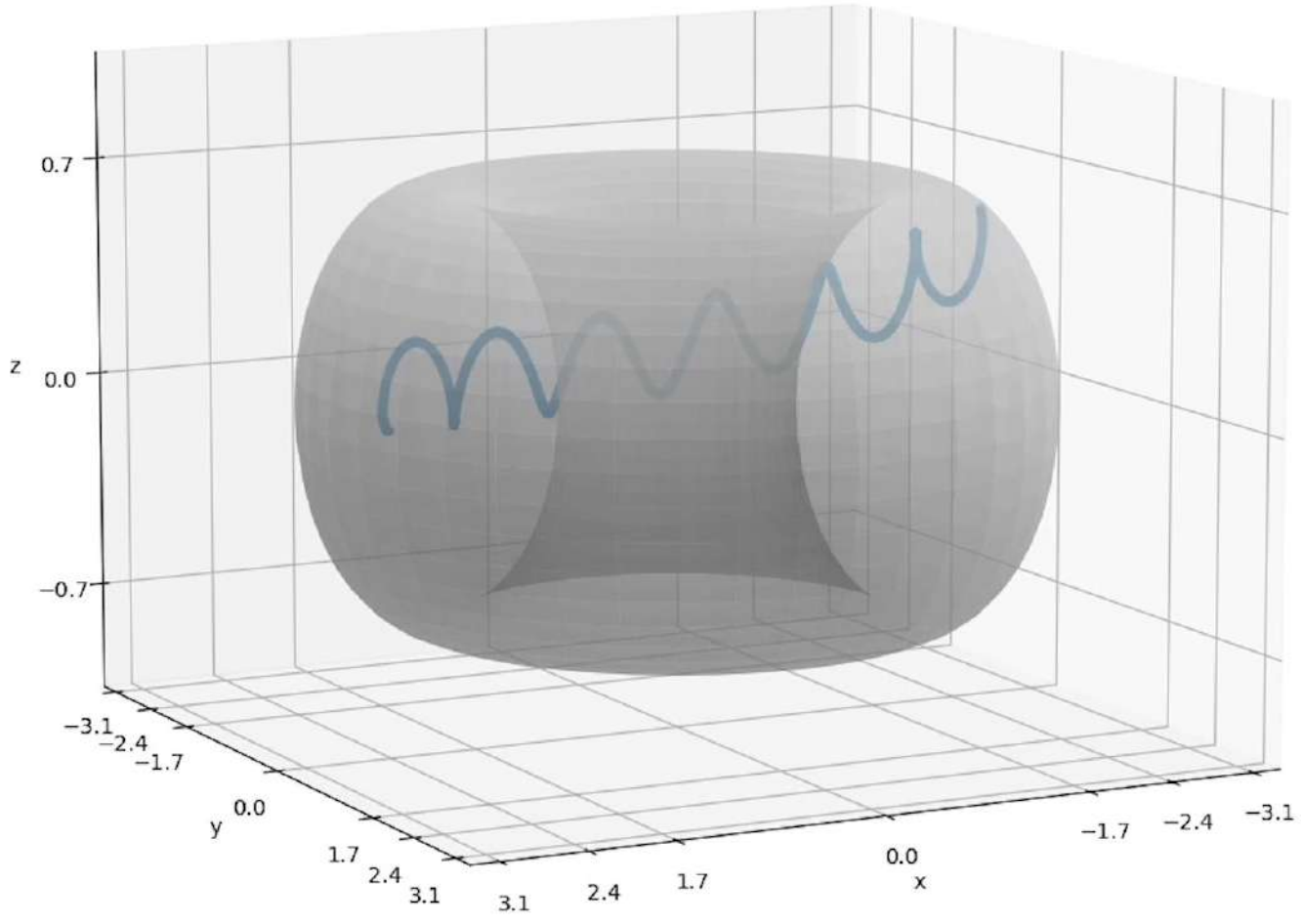

0.7
0.0
−0.7
z
−3.1
−2.4
−1.7
0.0
1.7
2.4
3.1
y
−3.1
−2.4
−1.7
0.0
1.7
2.4
3.1
x

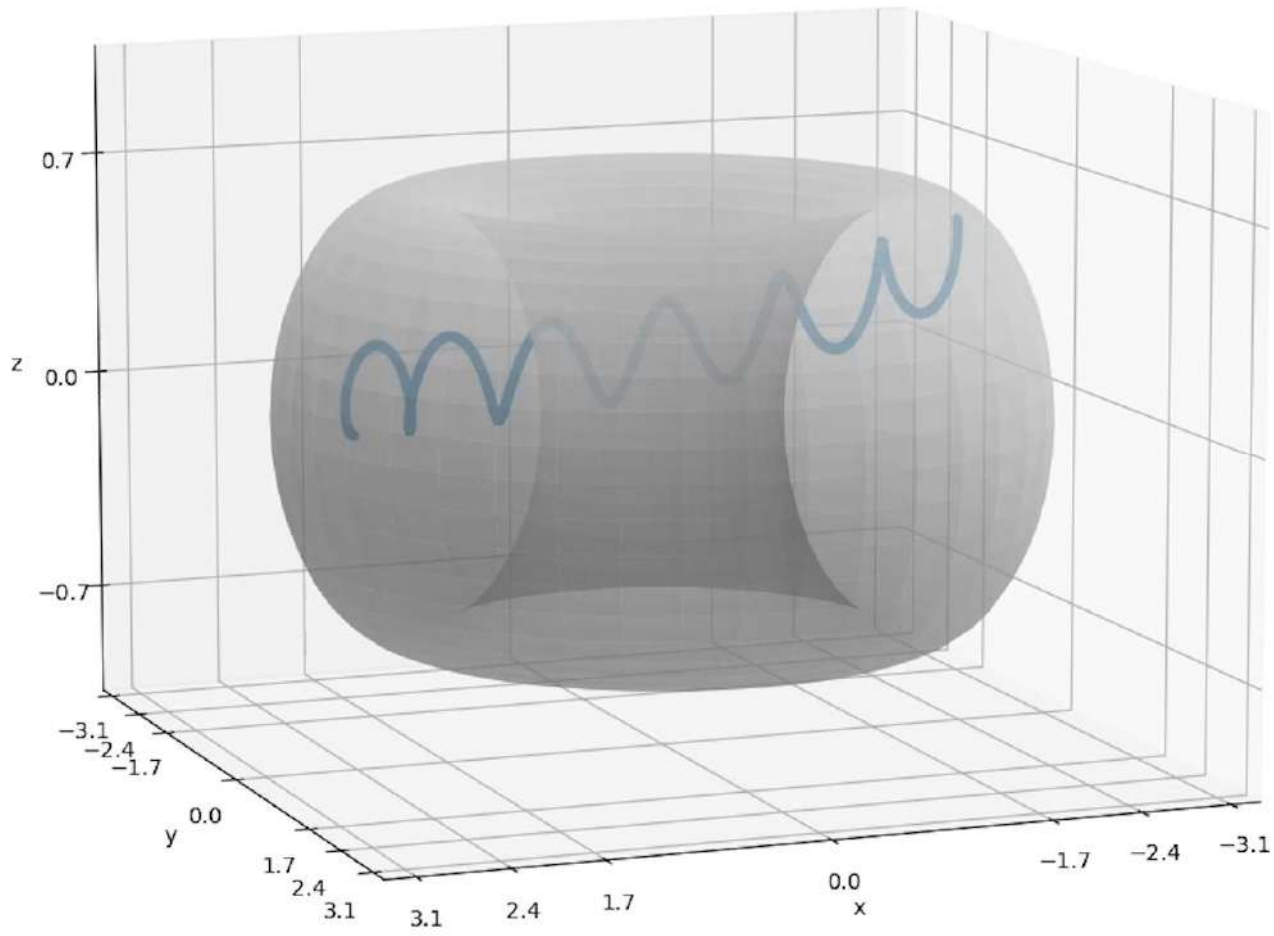

0.7
0.0
−0.7
z
−3.1
−2.4
−1.7
0.0
1.7
2.4
3.1
y
3.1
2.4
1.7
0.0
−1.7
−2.4
−3.1
x

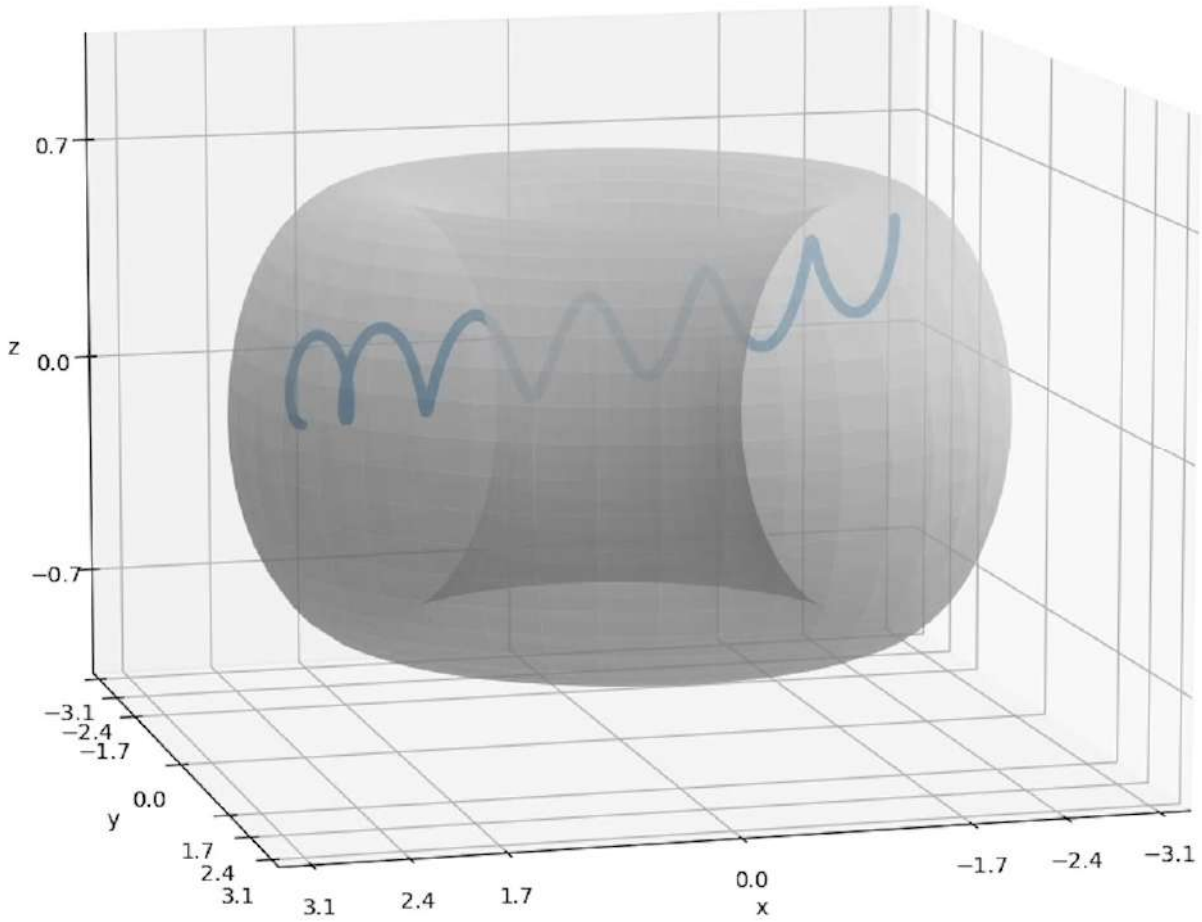

0.7
z
0.0
−0.7
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
3.1
2.4
1.7
0.0
x
−1.7
−2.4
−3.1

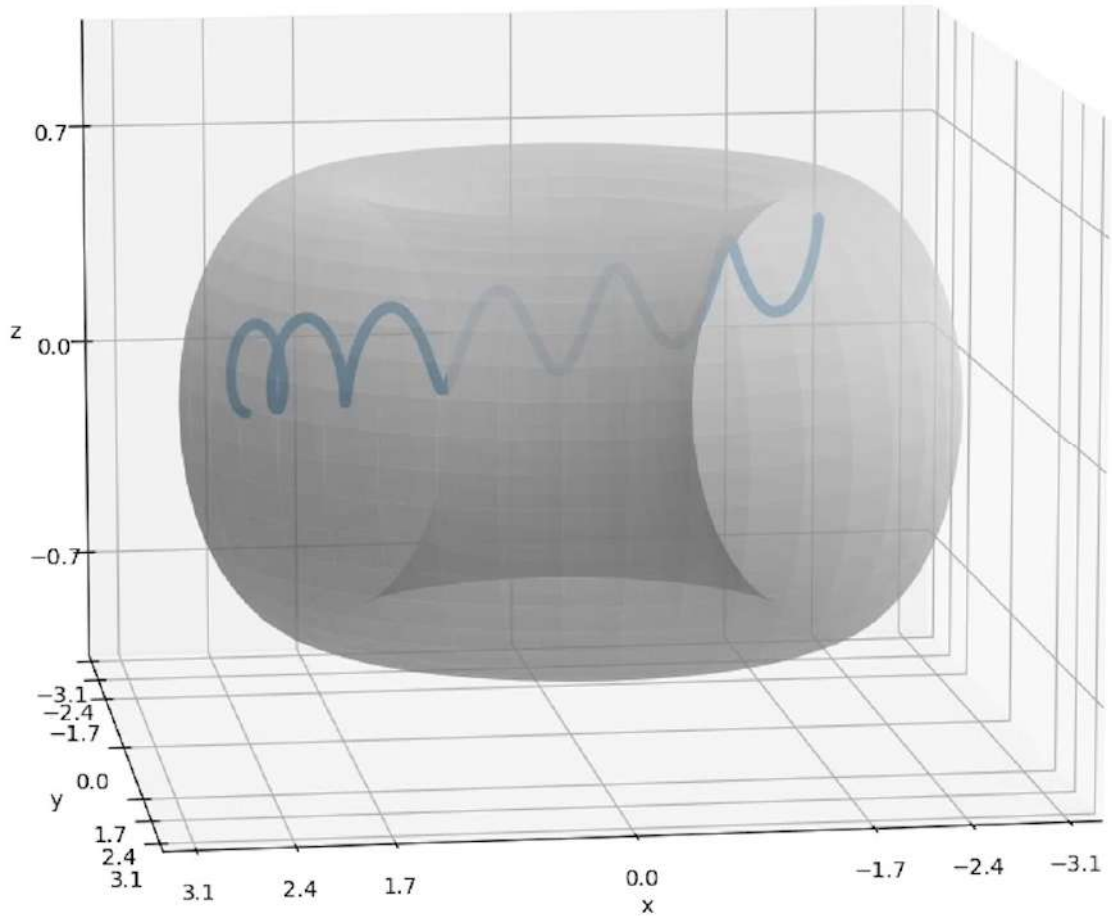

0.7
0.0
−0.7
z
−3.1
−2.4
−1.7
0.0
1.7
2.4
3.1
y
3.1
2.4
1.7
0.0
−1.7
−2.4
−3.1
x

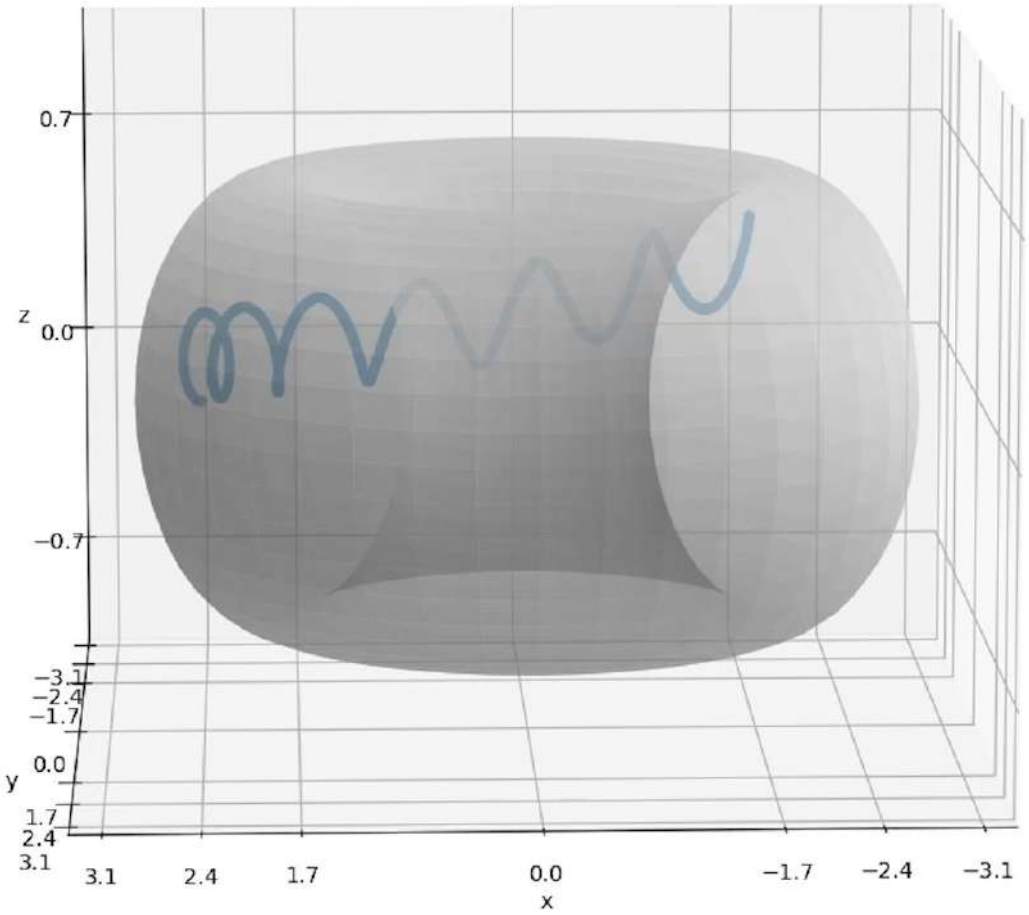

0.7
0.0
−0.7
z
−3.1
−2.4
−1.7
0.0
1.7
2.4
3.1
y
3.1
2.4
1.7
0.0
−1.7
−2.4
−3.1
x

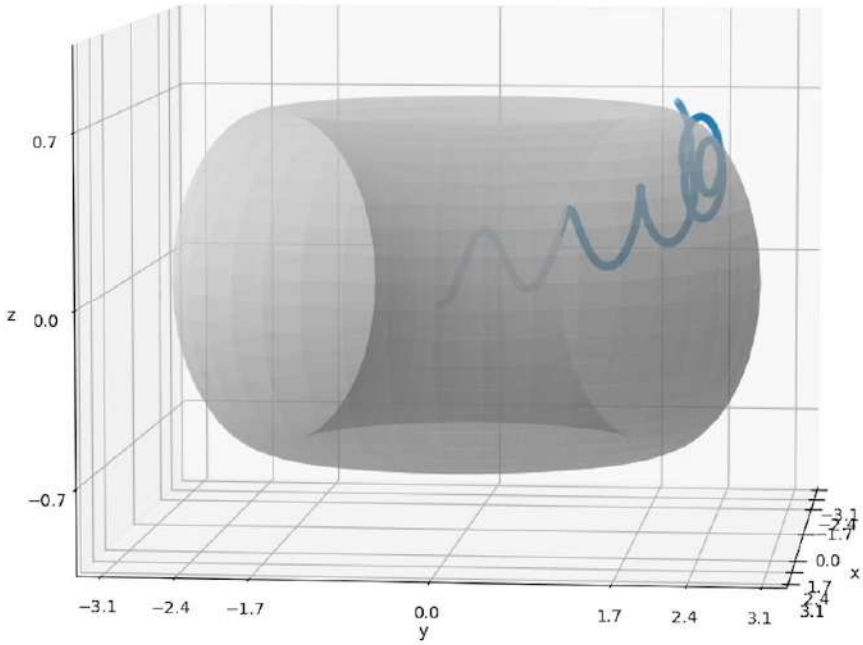

0.7
0.0
−0.7
z
−3.1
−2.4
−1.7
0.0
1.7
2.4
3.1
y
3.1
2.4
1.7
0.0
−1.7
−2.4
−3.1
x

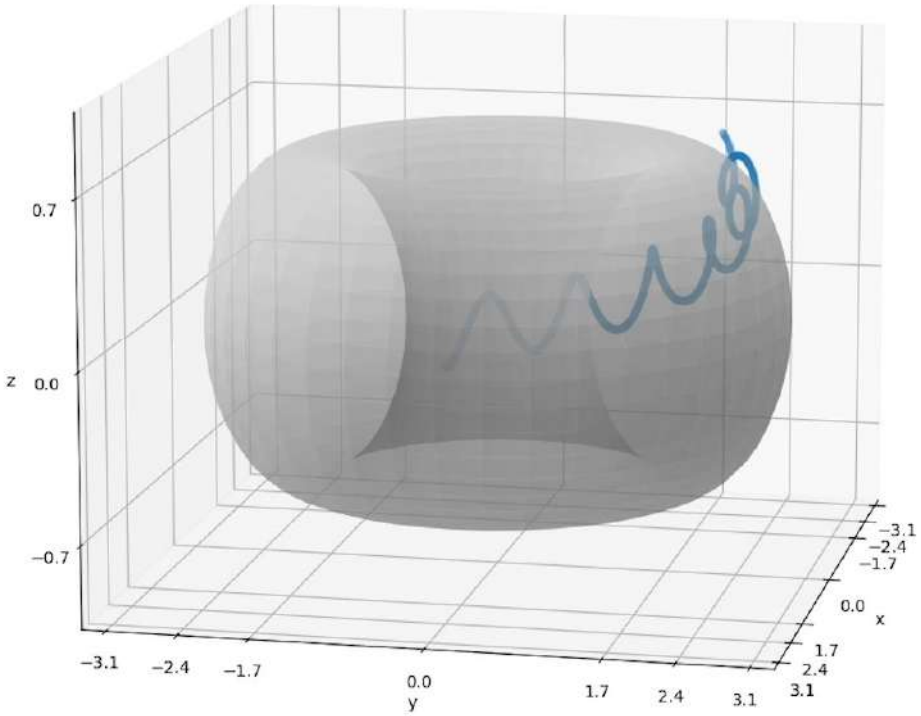

0.7
0.0
−0.7
z
−3.1
−2.4
−1.7
0.0
1.7
2.4
3.1
y
3.1
2.4
1.7
0.0
−1.7
−2.4
−3.1
x

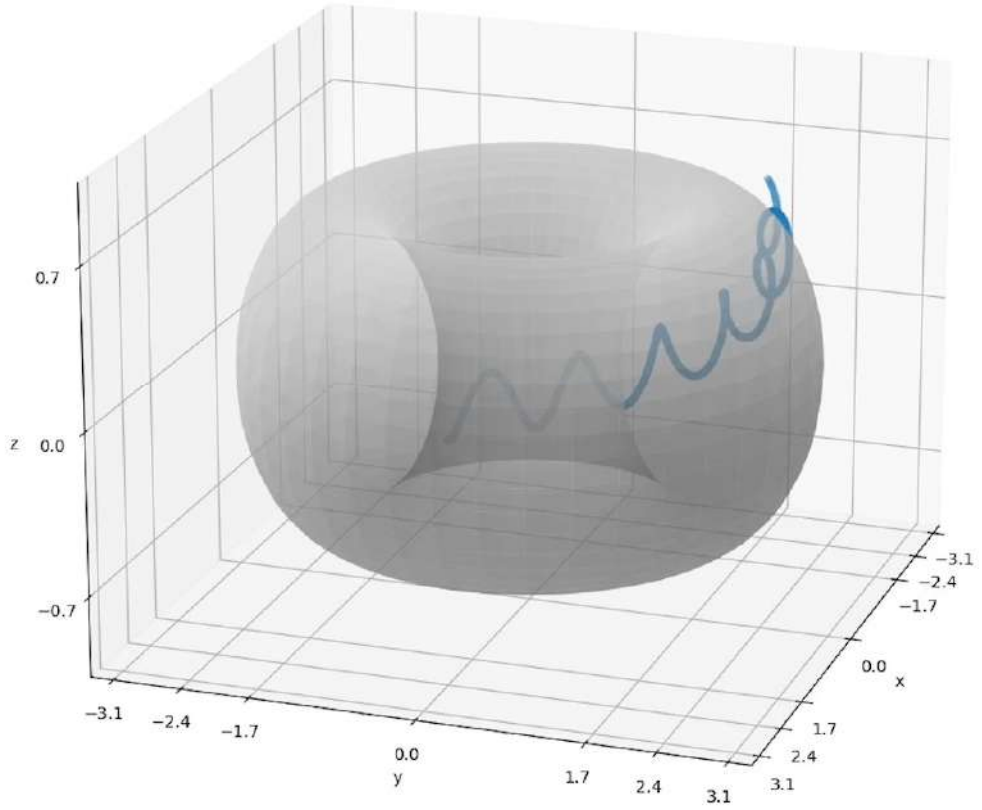

0.7
z
0.0
−0.7
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
3.1
2.4
1.7
0.0
x
−1.7
−2.4
−3.1

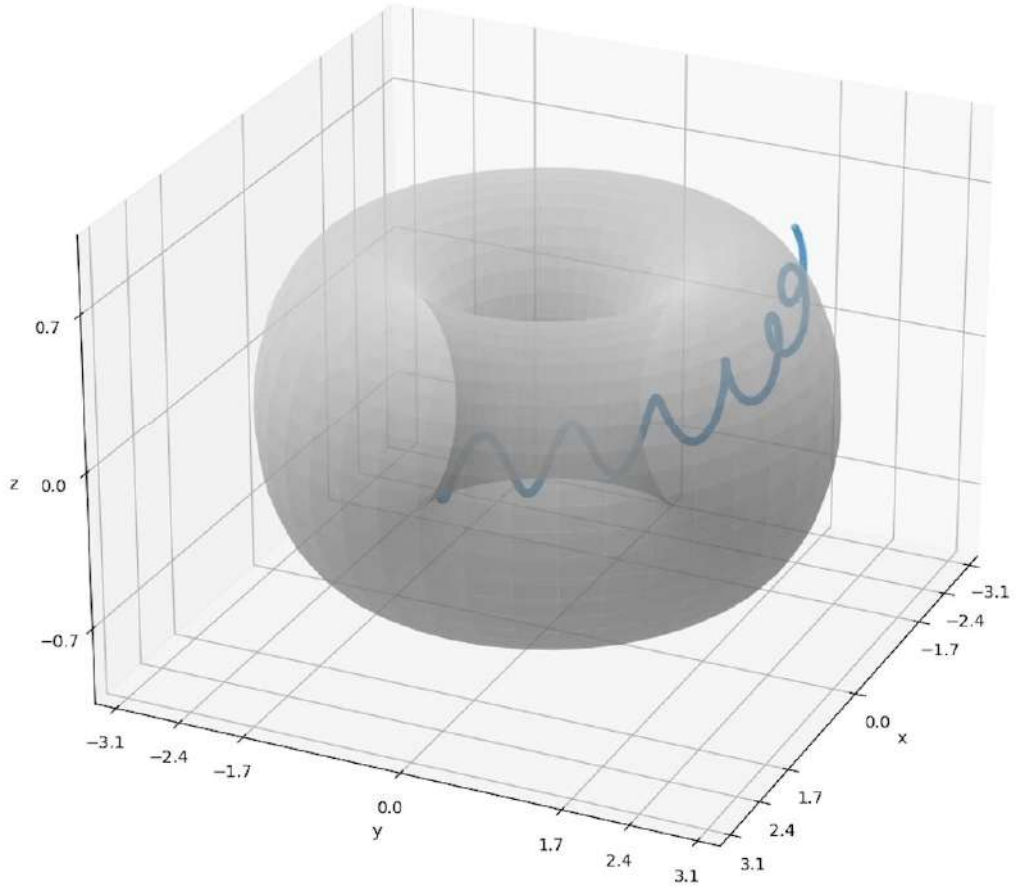

0.7
0.0
−0.7
z
−3.1
−2.4
−1.7
0.0
1.7
2.4
3.1
y
3.1
2.4
1.7
0.0
−1.7
−2.4
−3.1
x

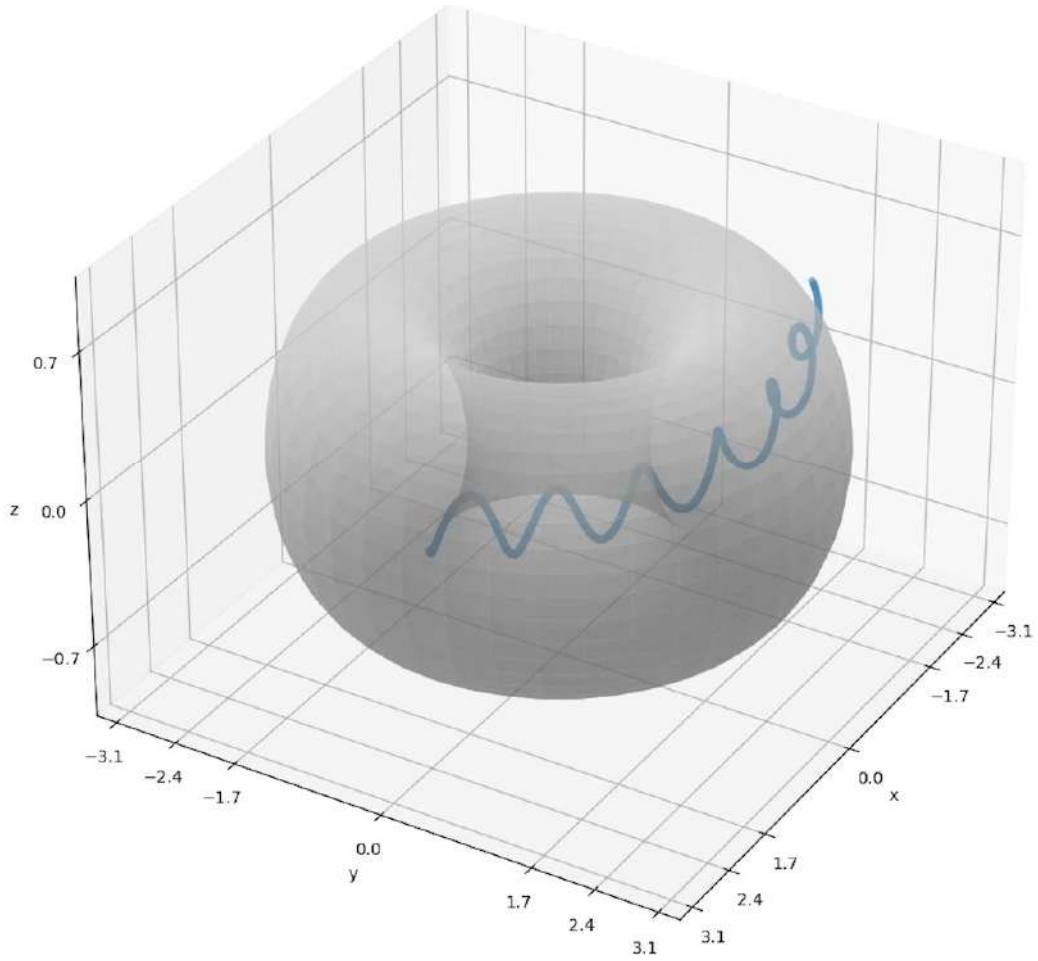

0.7
z
0.0
−0.7
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
3.1
2.4
1.7
0.0
−1.7
−2.4
−3.1
x

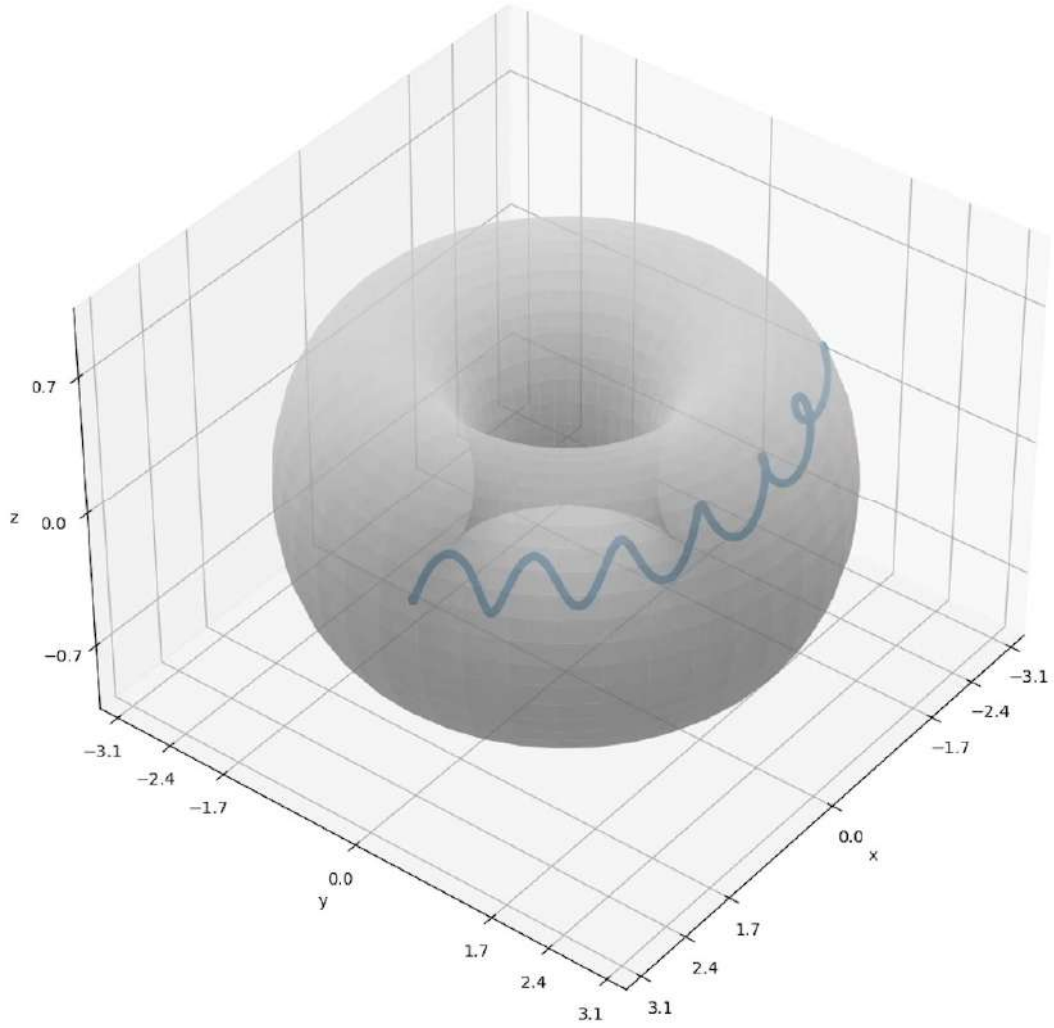

0.7
z
0.0
−0.7
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
−3.1
−2.4
−1.7
0.0
x
1.7
2.4
3.1

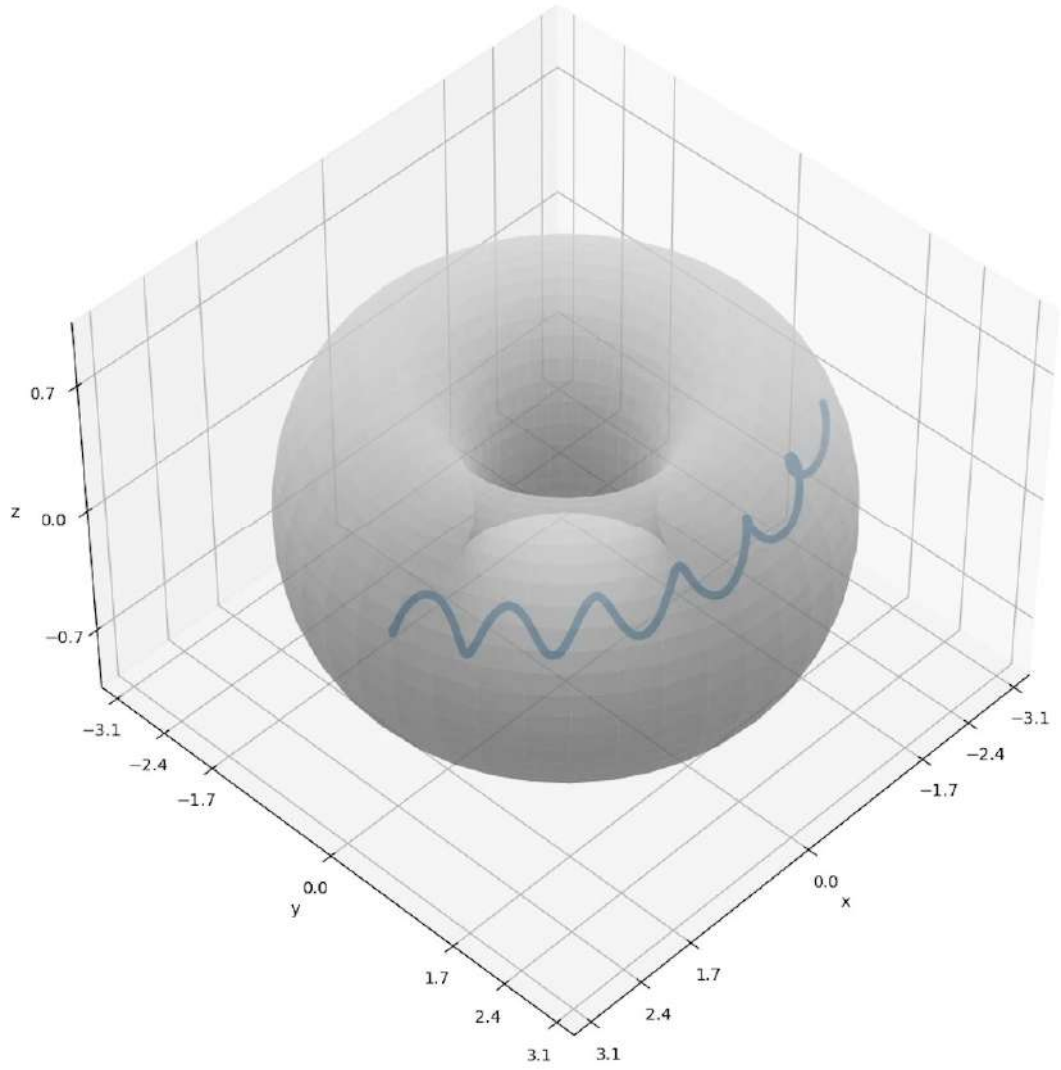

0.7
z 0.0
−0.7
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
−3.1
−2.4
−1.7
0.0
x
1.7
2.4
3.1

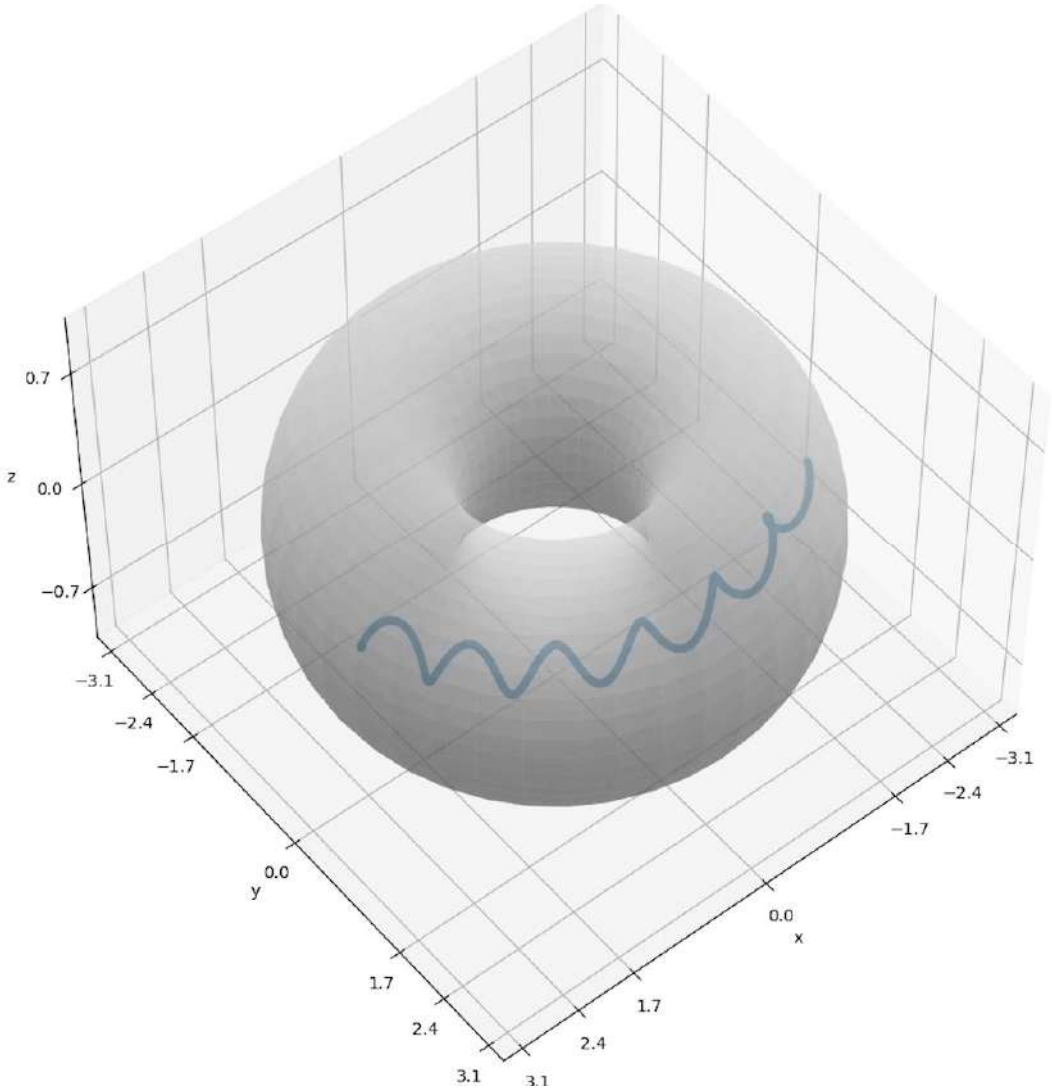

0.7
z 0.0
−0.7
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
−3.1
−2.4
−1.7
0.0
x
1.7
2.4
3.1

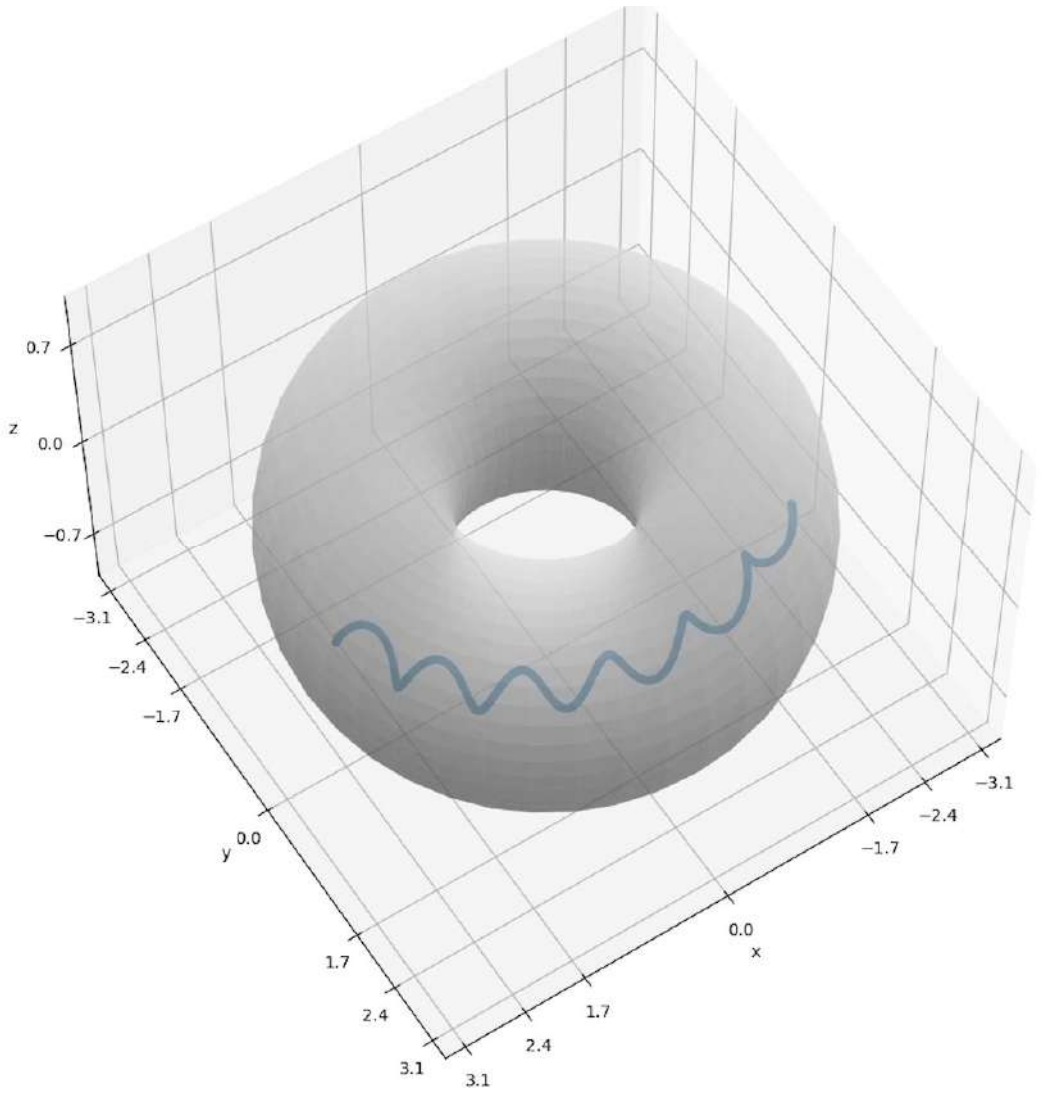

0.7
z 0.0
−0.7
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
−3.1
−2.4
−1.7
0.0
x
1.7
2.4
3.1

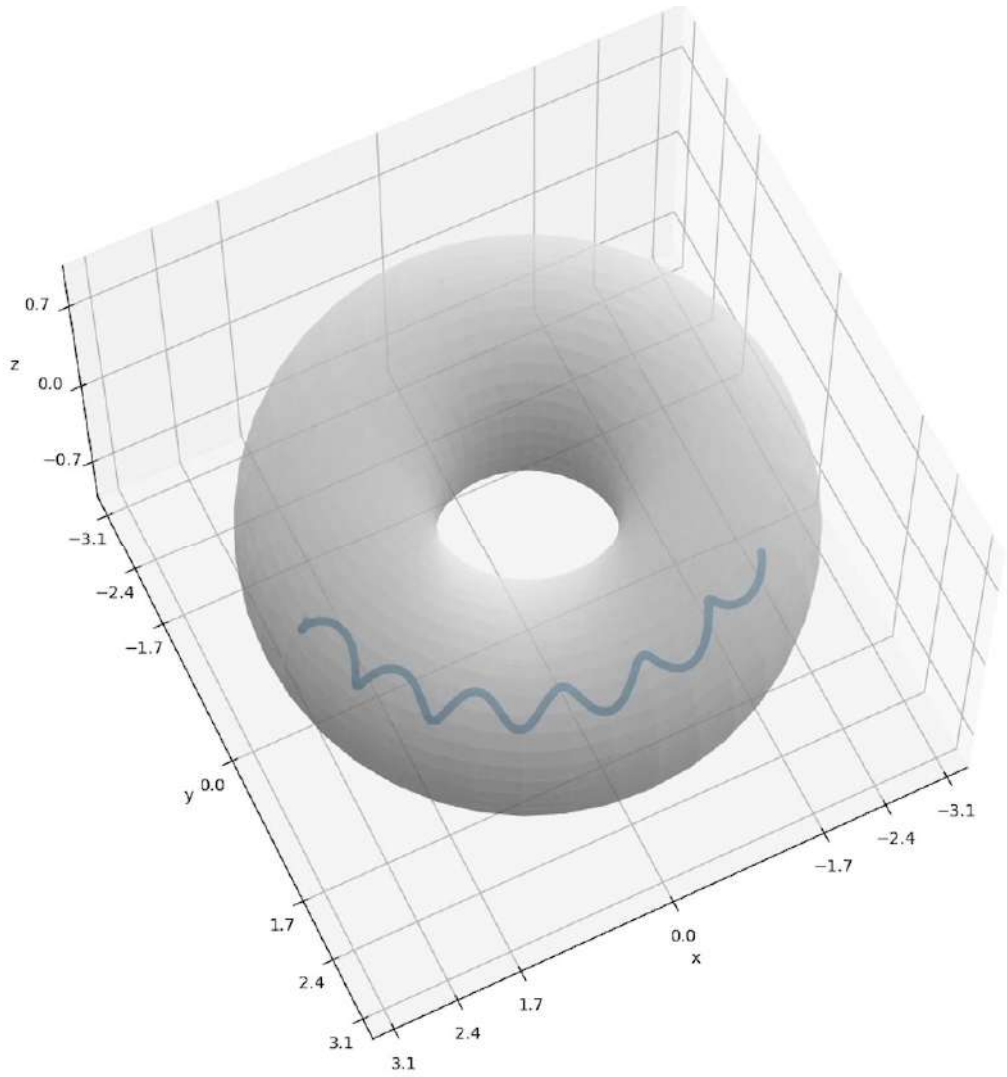

0.7
z 0.0
−0.7
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
−3.1
−2.4
−1.7
0.0
x
1.7
2.4
3.1

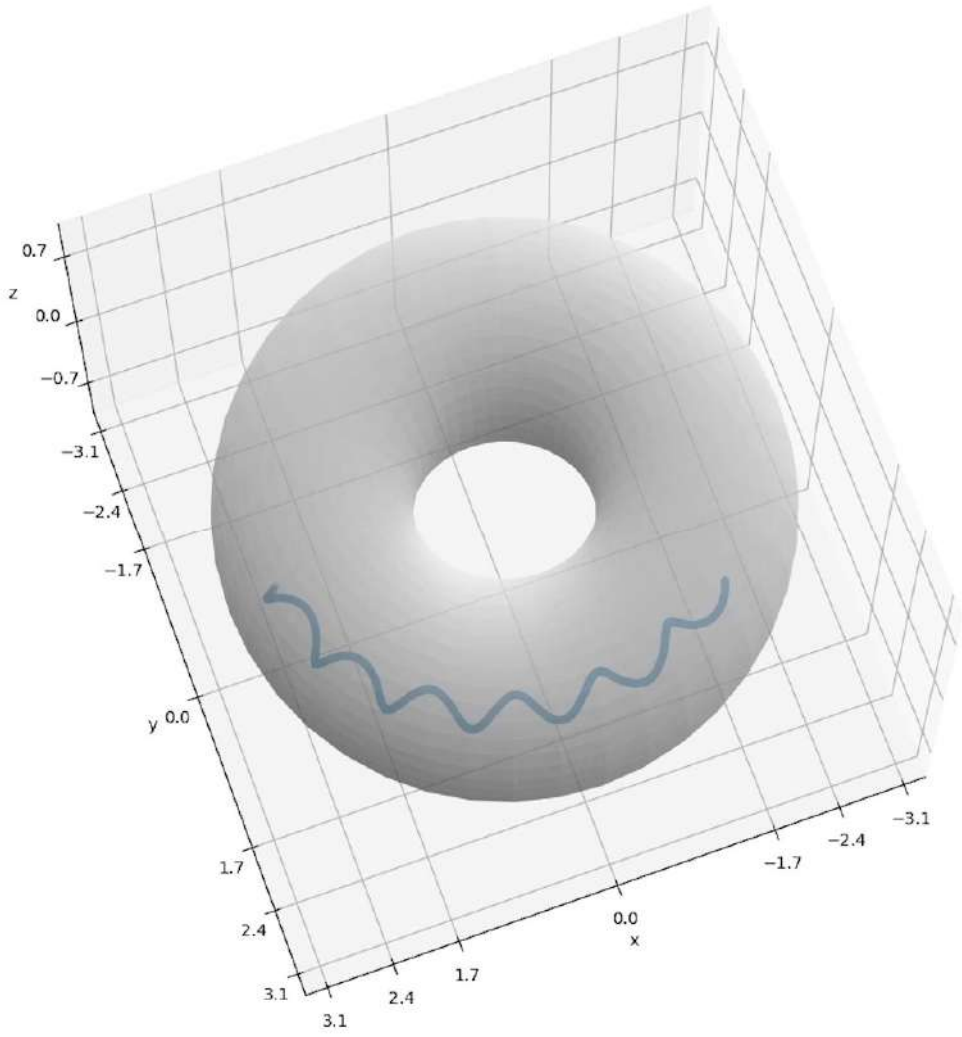

0.7
0.0
−0.7
z
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
−3.1
−2.4
−1.7
0.0
x
1.7
2.4
3.1

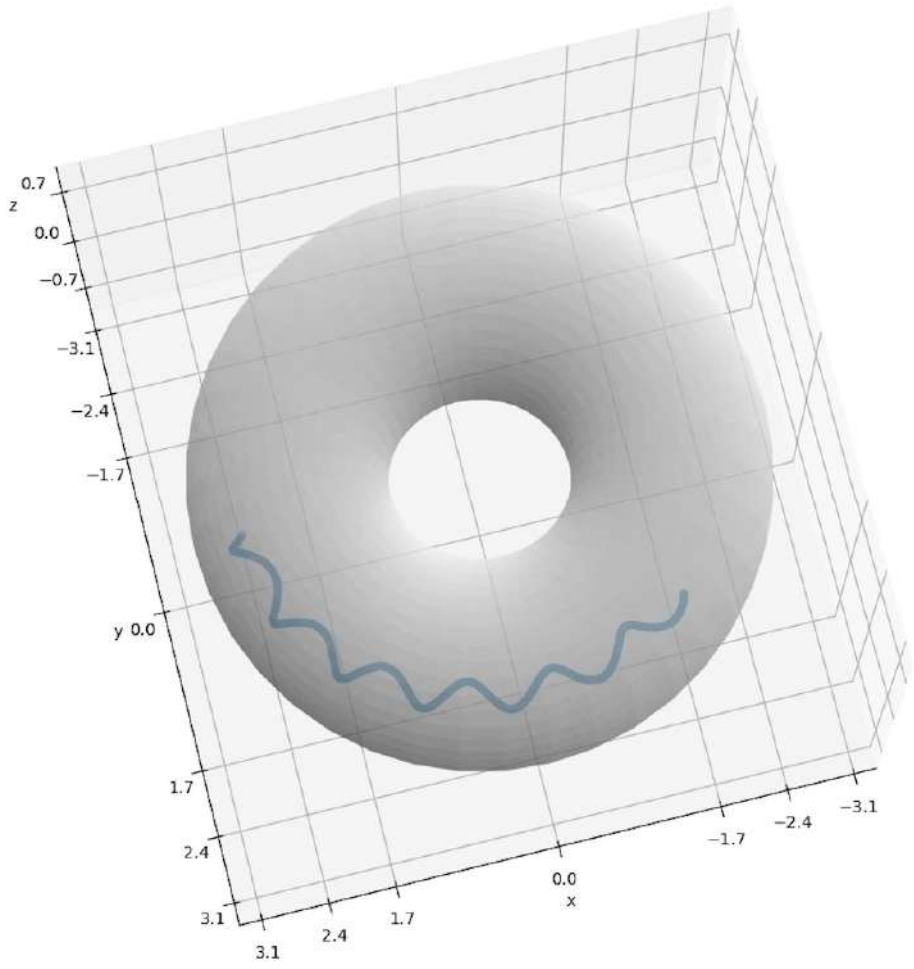

0.7
0.0
−0.7
z
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
3.1
2.4
1.7
0.0
x
−1.7
−2.4
−3.1

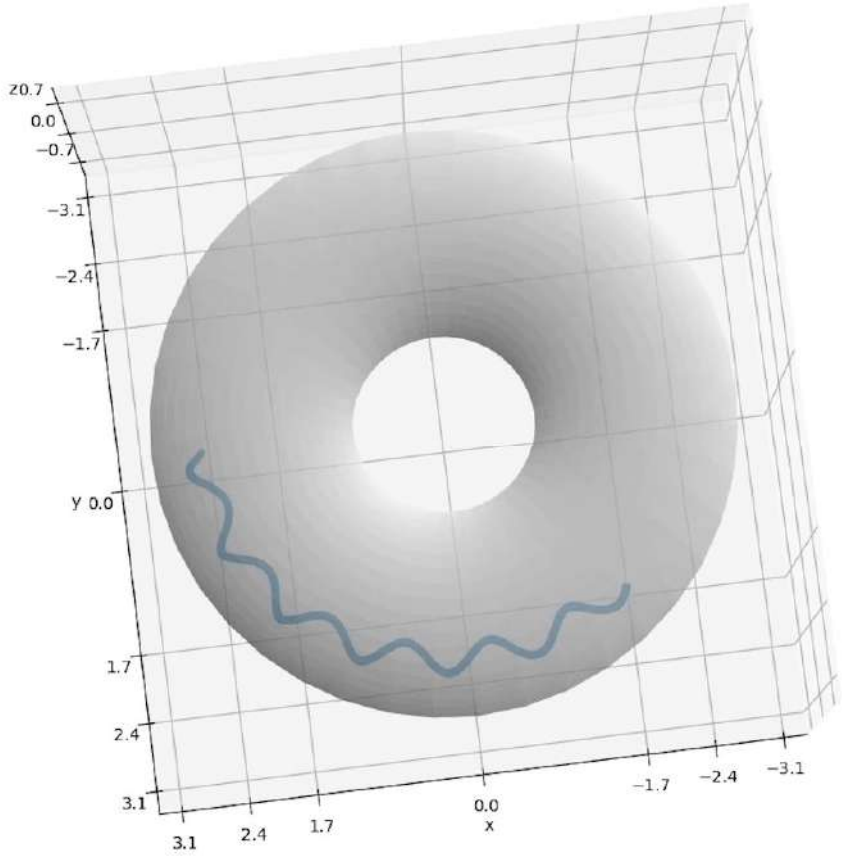

0.7
0.0
−0.7
z
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
−3.1
−2.4
−1.7
0.0
x
1.7
2.4
3.1

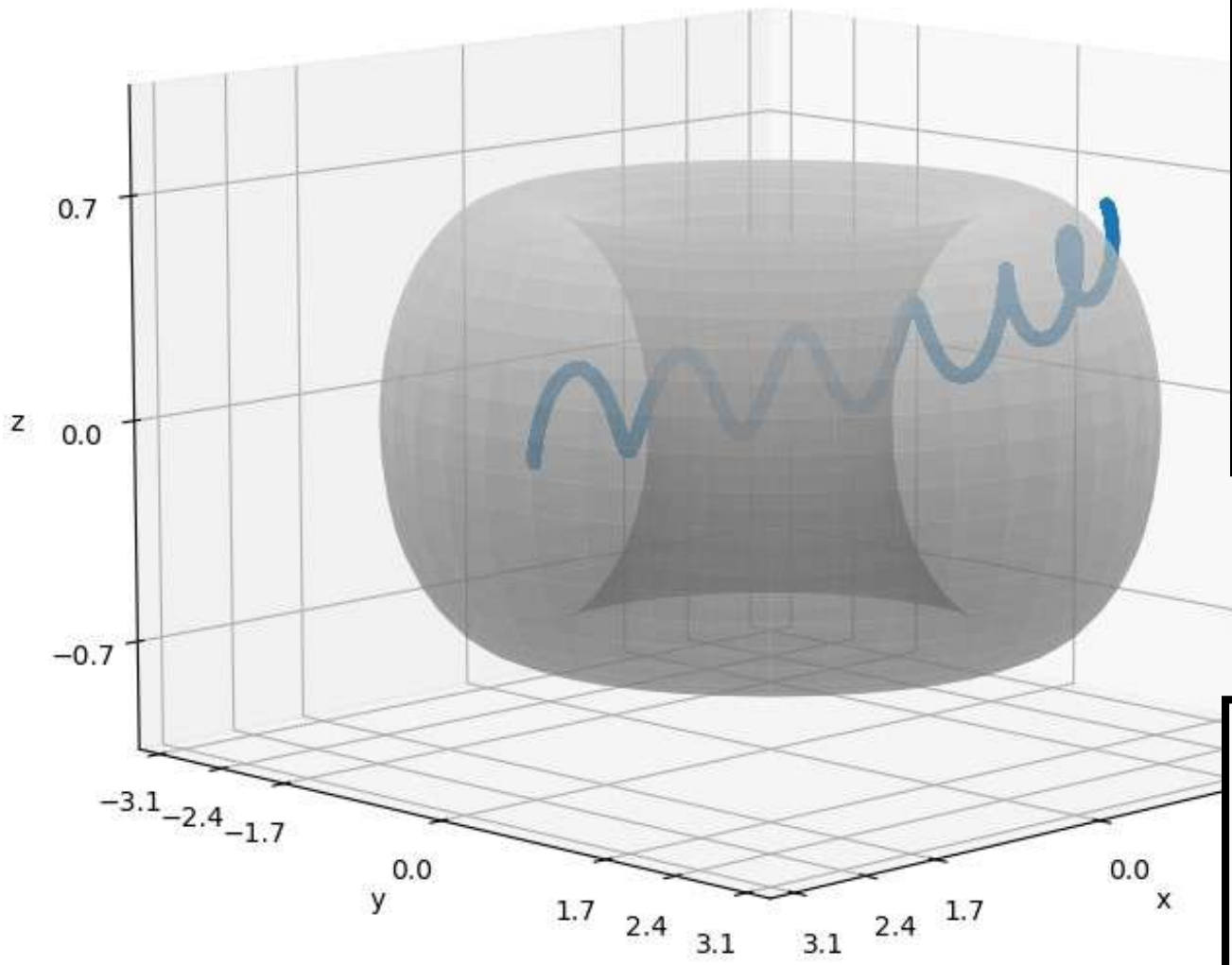

0.7
0.0
−0.7
z
−3.1
−2.4
−1.7
0.0
y
1.7
2.4
3.1
3.1
2.4
1.7
0.0
x
−1.7
−2.4
−3.1

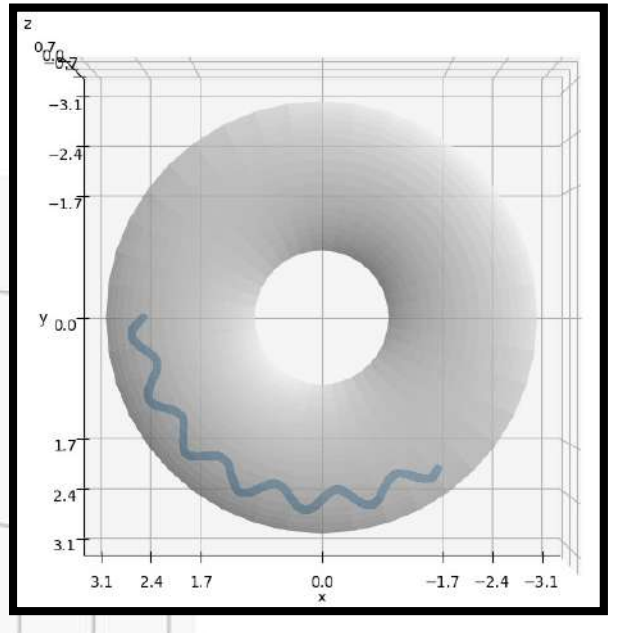

0.7
0.0
z
−0.7
−3.1
−2.4
−1.7
y 0.0
1.7
2.4
3.1
3.1
2.4
1.7
0.0
x
−1.7
−2.4
−3.1

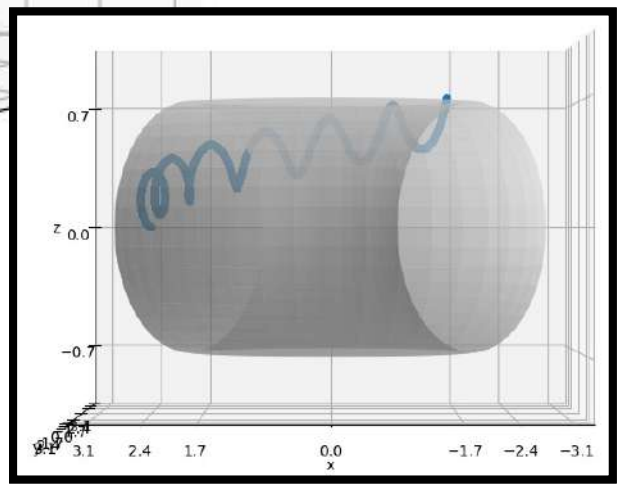

0.7
z
0.0
−0.7
−3.1
−2.4
−1.7
y 0.0
1.7
2.4
3.1
3.1
2.4
1.7
0.0
x
−1.7
−2.4
−3.1


0.7
z
0.0
−0.7
−3.1
−2.4
−1.7
y 0.0
1.7
2.4
3.1
3.1
2.4
1.7
0.0
x
−1.7
−2.4
−3.1


z0.7
0.0
−0.7
−3.1
−2.4
−1.7
y 0.0
1.7
2.4
3.1
3.1
2.4
1.7
0.0
x
−1.7
−2.4
−3.1

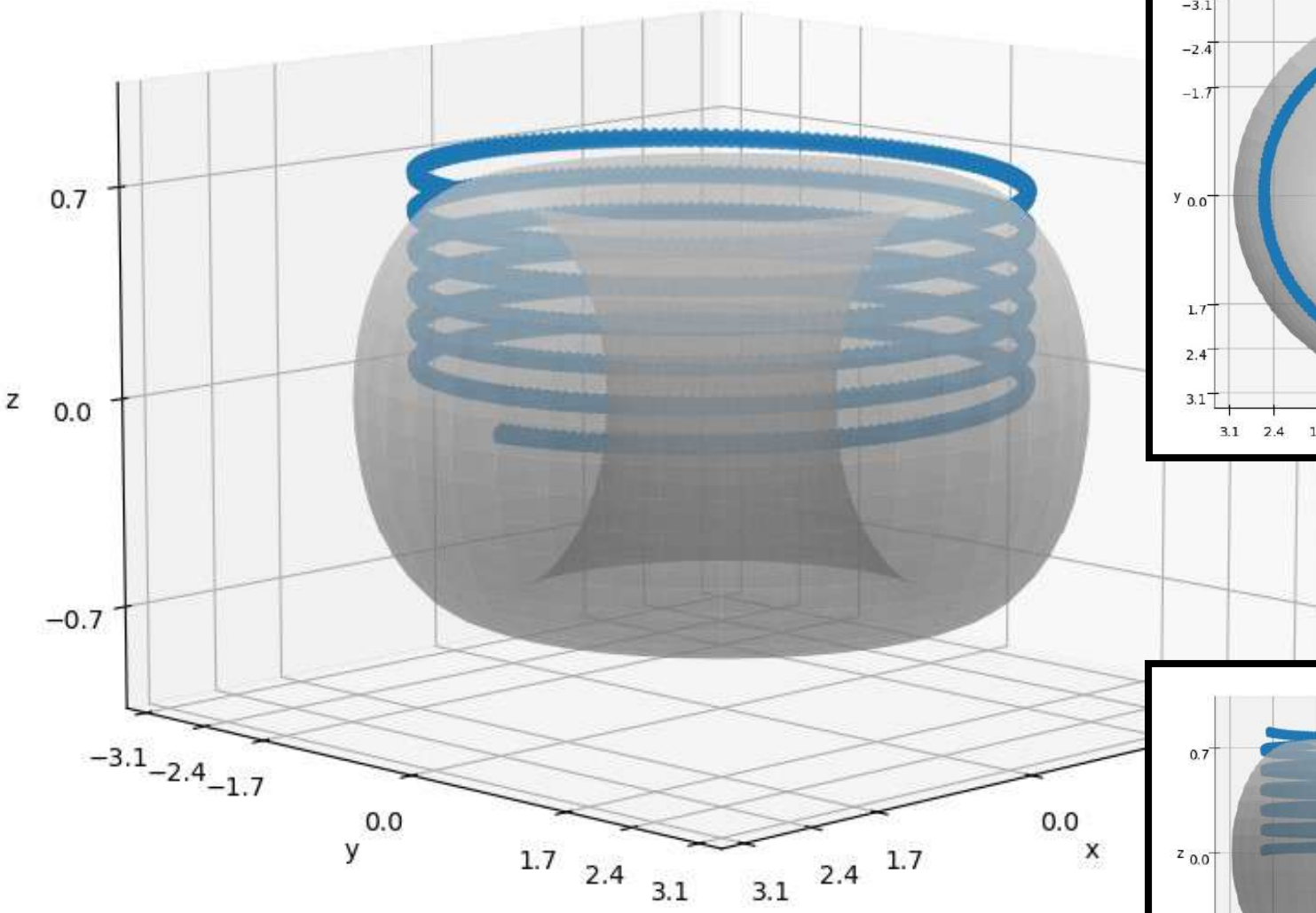


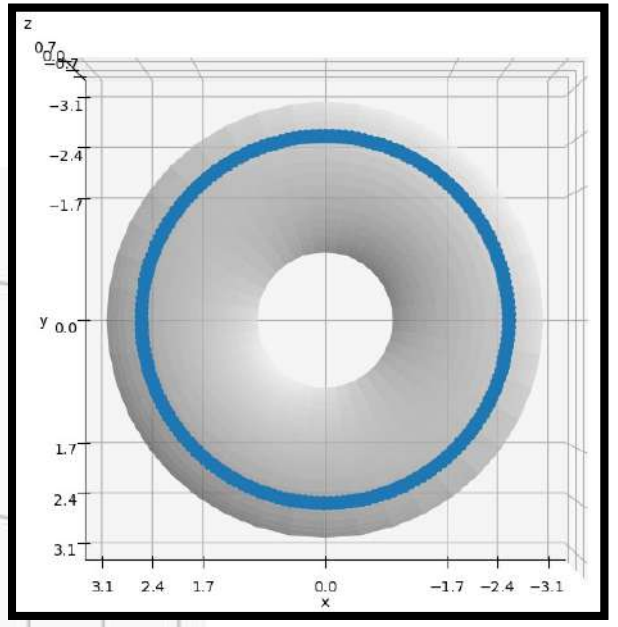


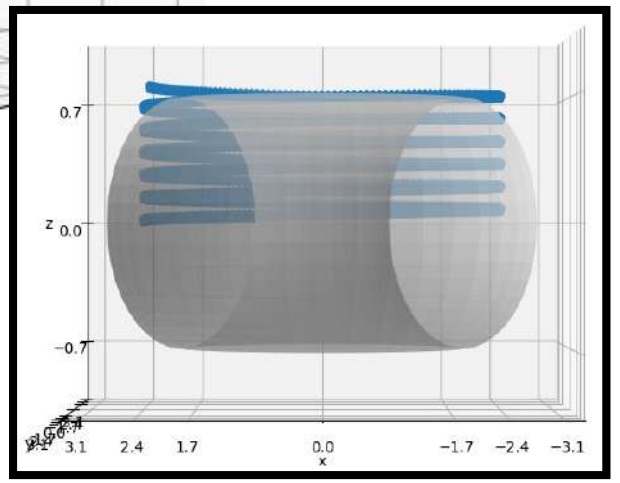


Temps simulé: $5,4\,.\,10^{-11}$ s

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

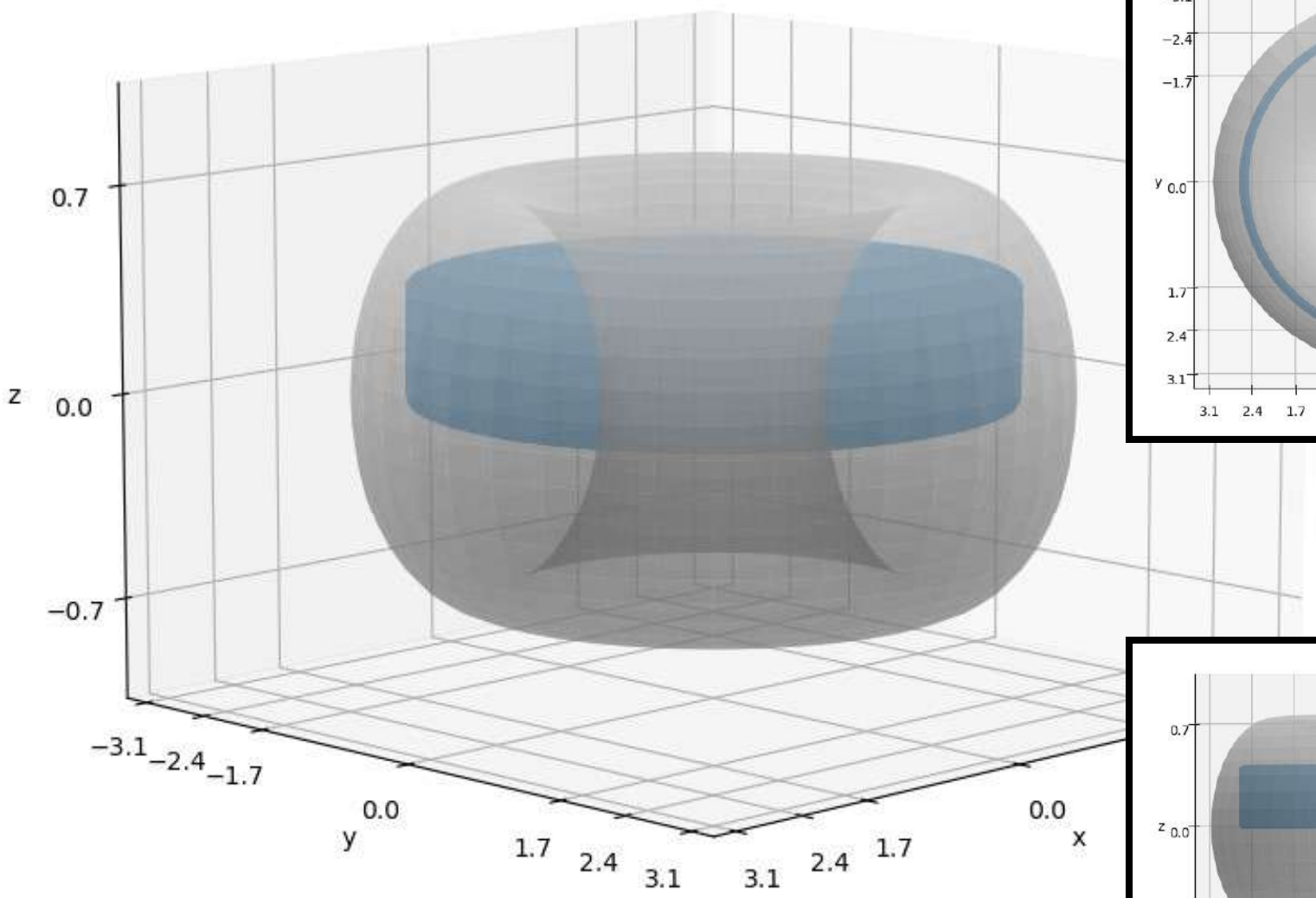


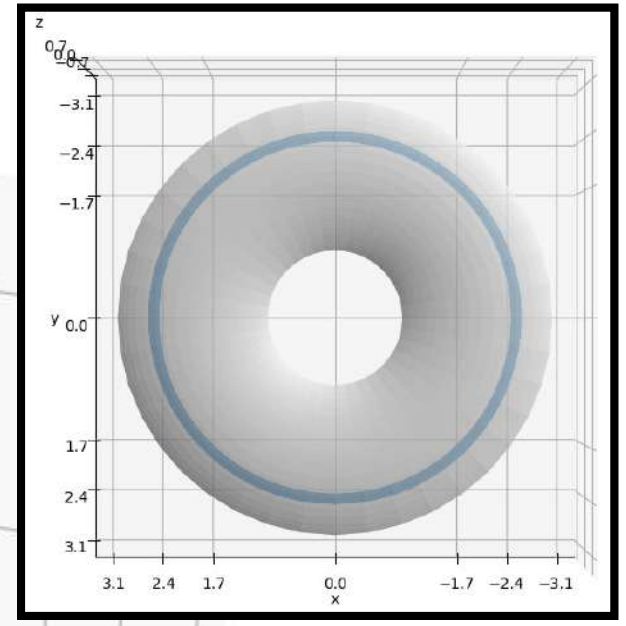


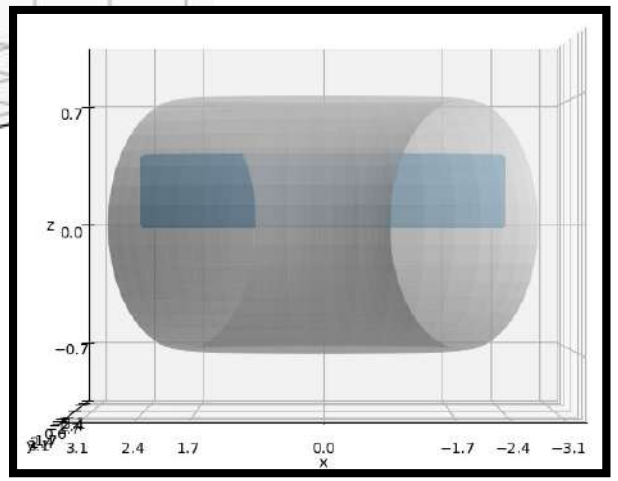


Temps simulé: 10^{-8} s

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

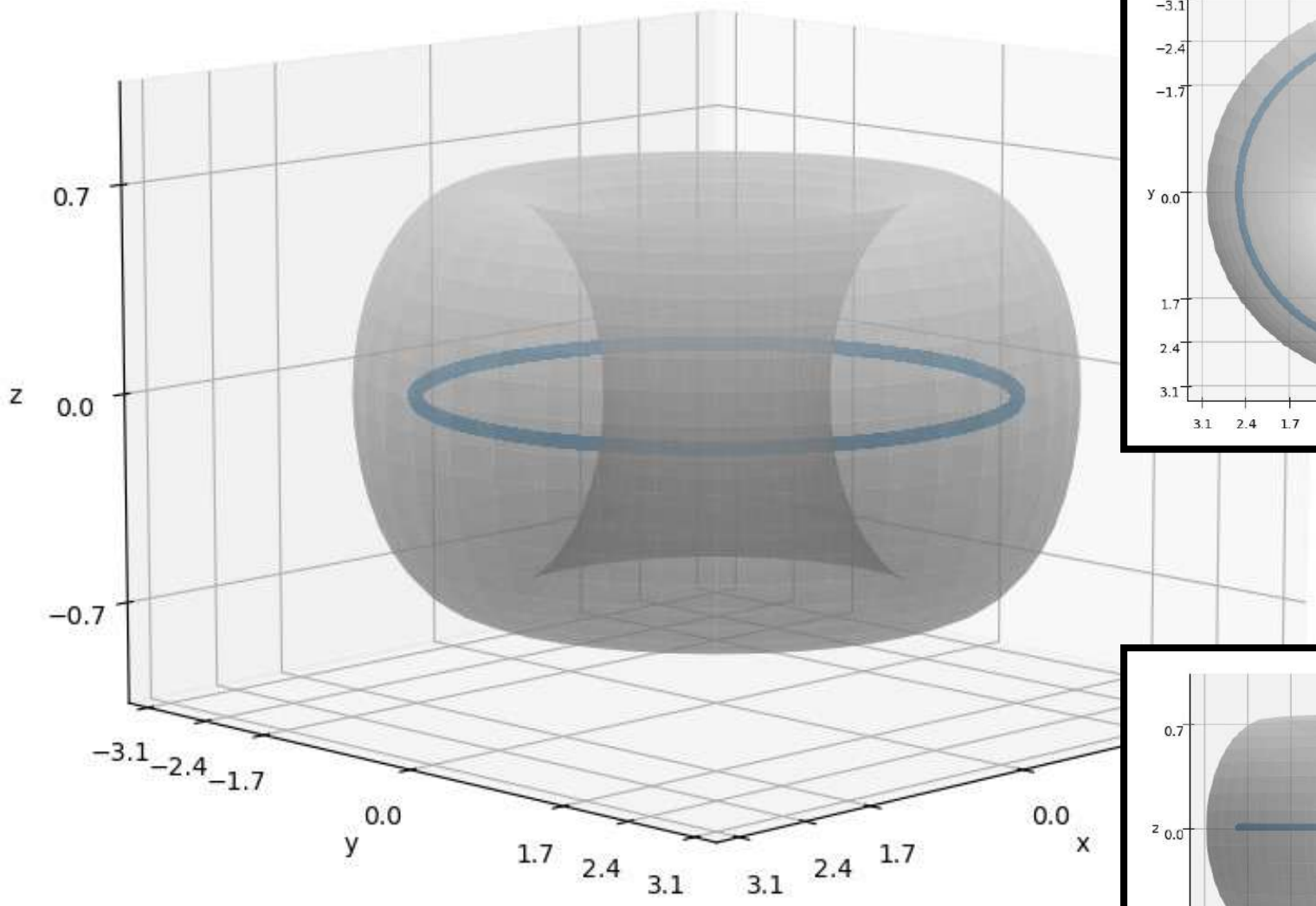


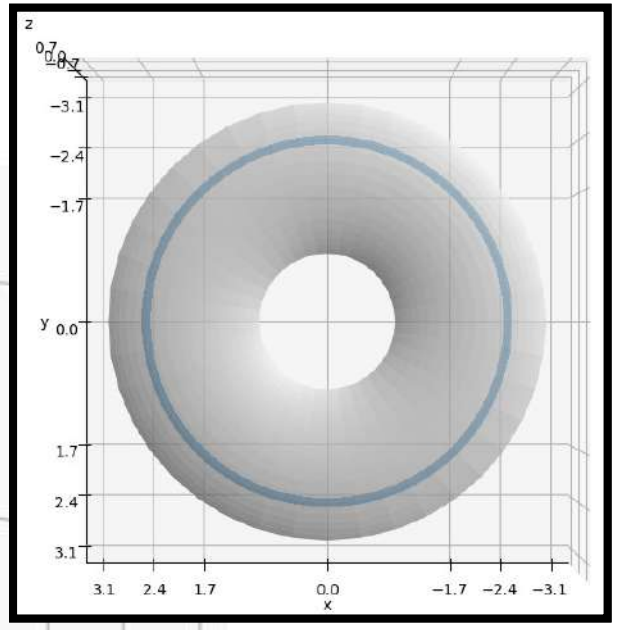


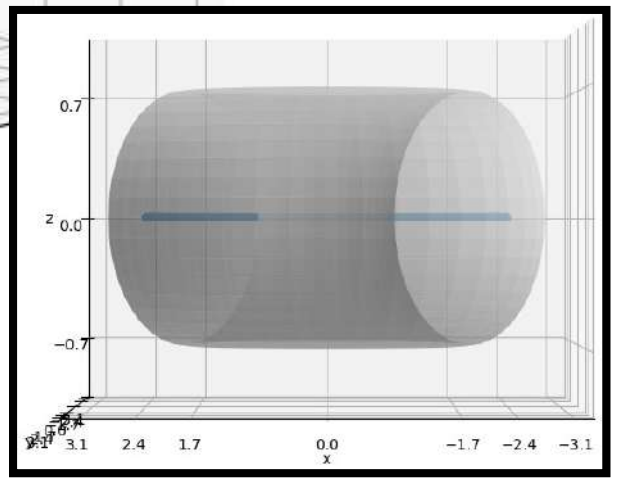


Temps simulé: 10^{-4} s

Durée de confinement estimée: 2.10^{-4} s



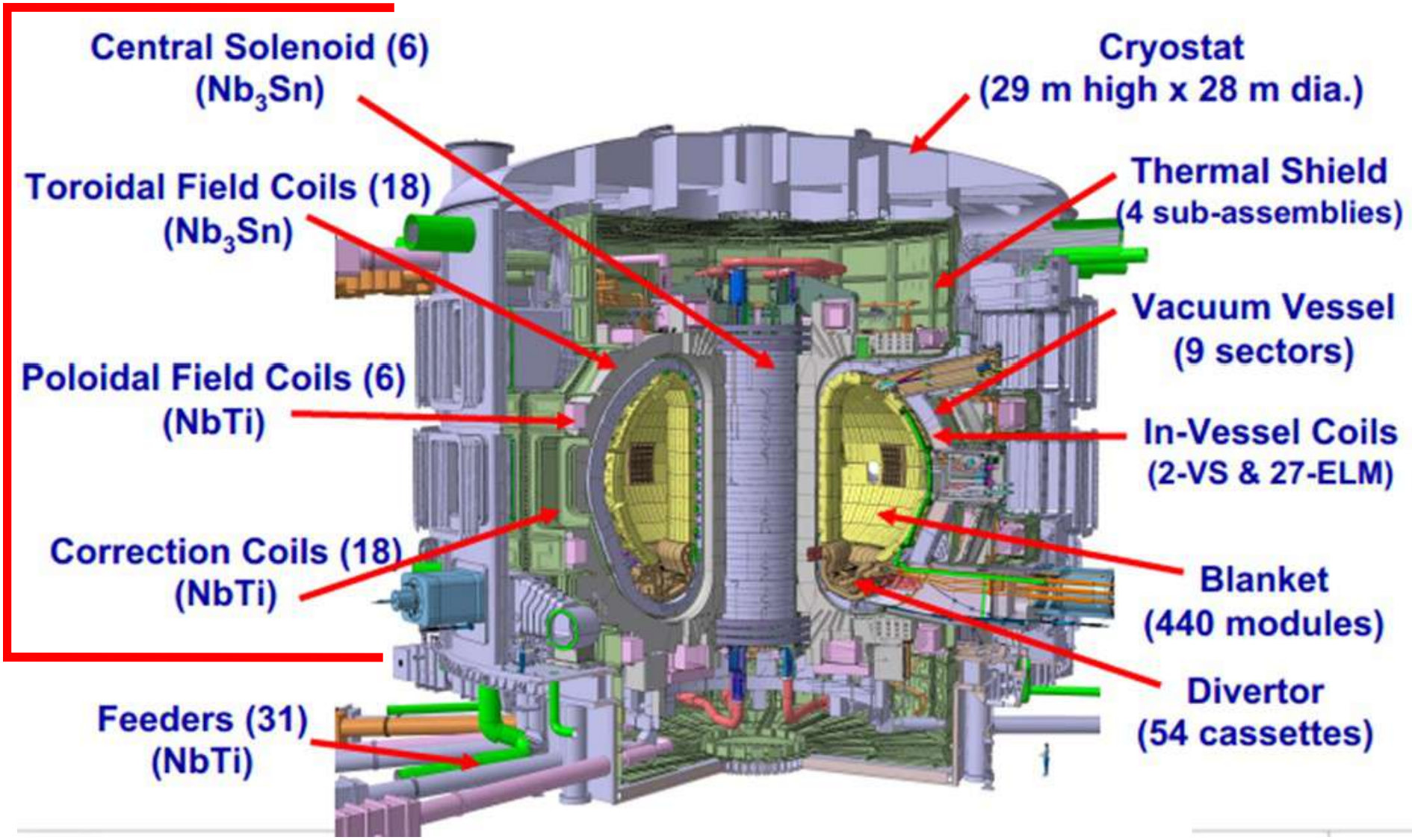




Temps simulé: 10^{-4} s

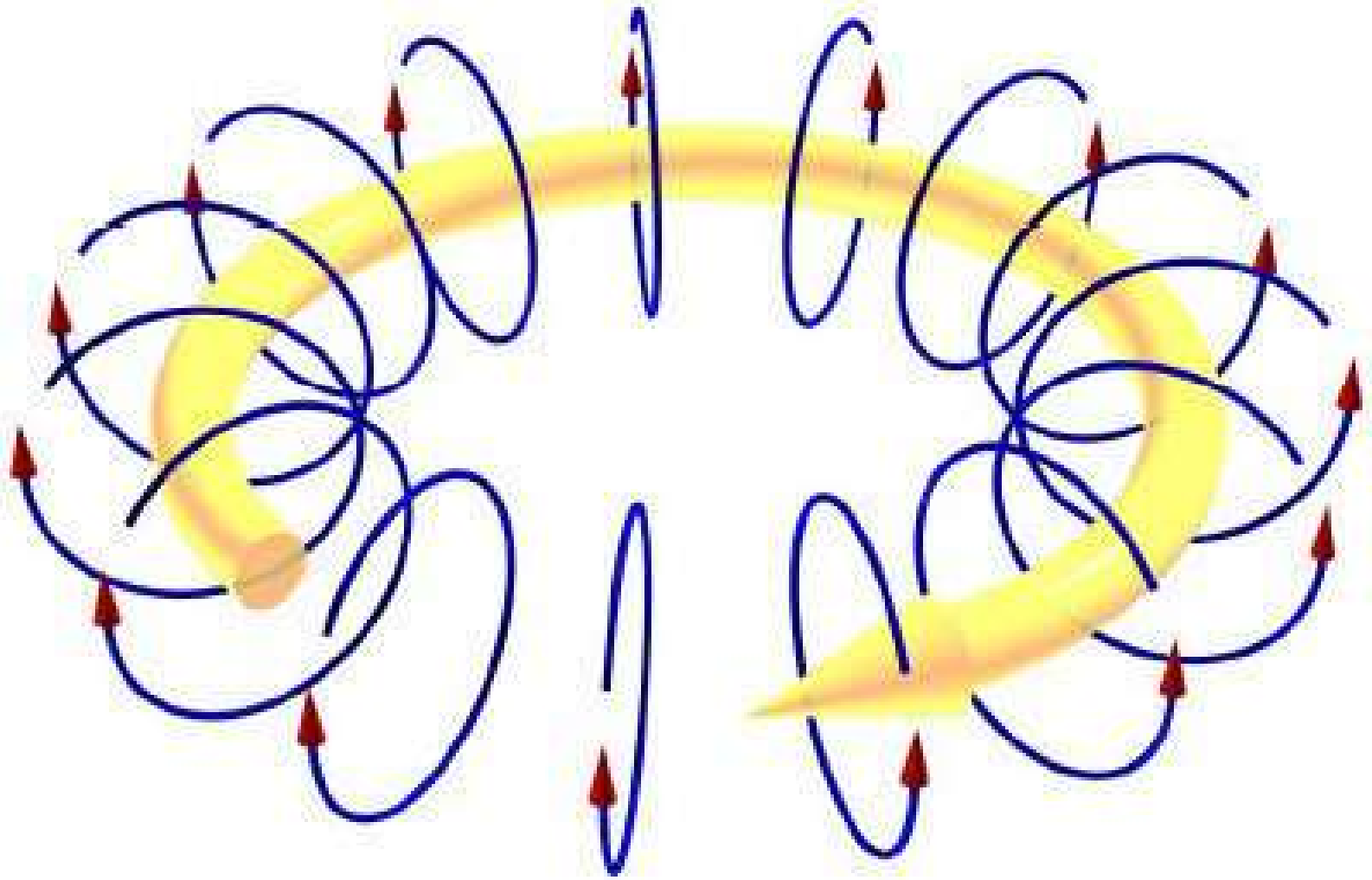
Durée de confinement estimée: $1,17$ s

III: Amélioration du Modèle

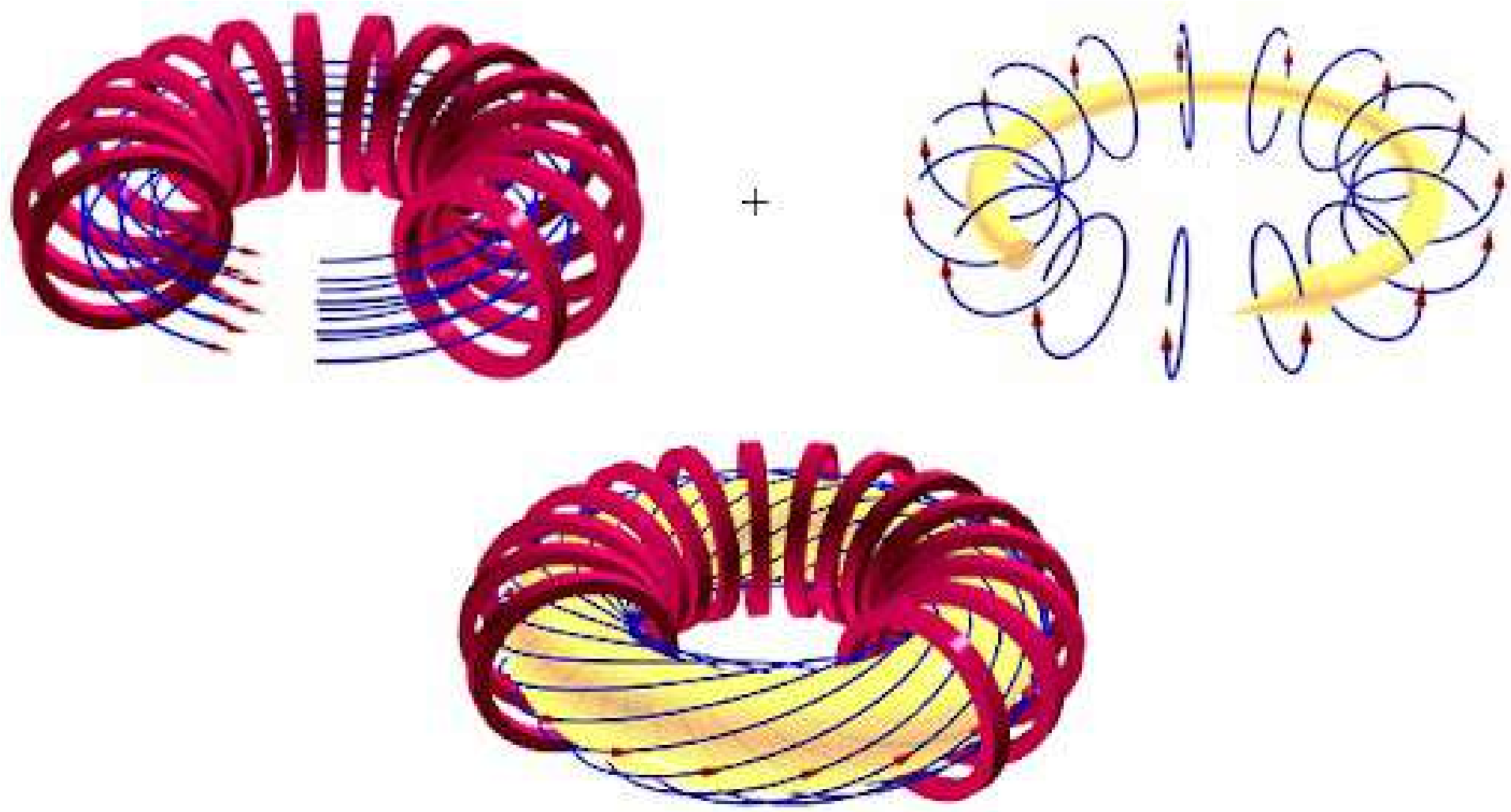


Source: iter.org

Champ Magnétique Poloïdal

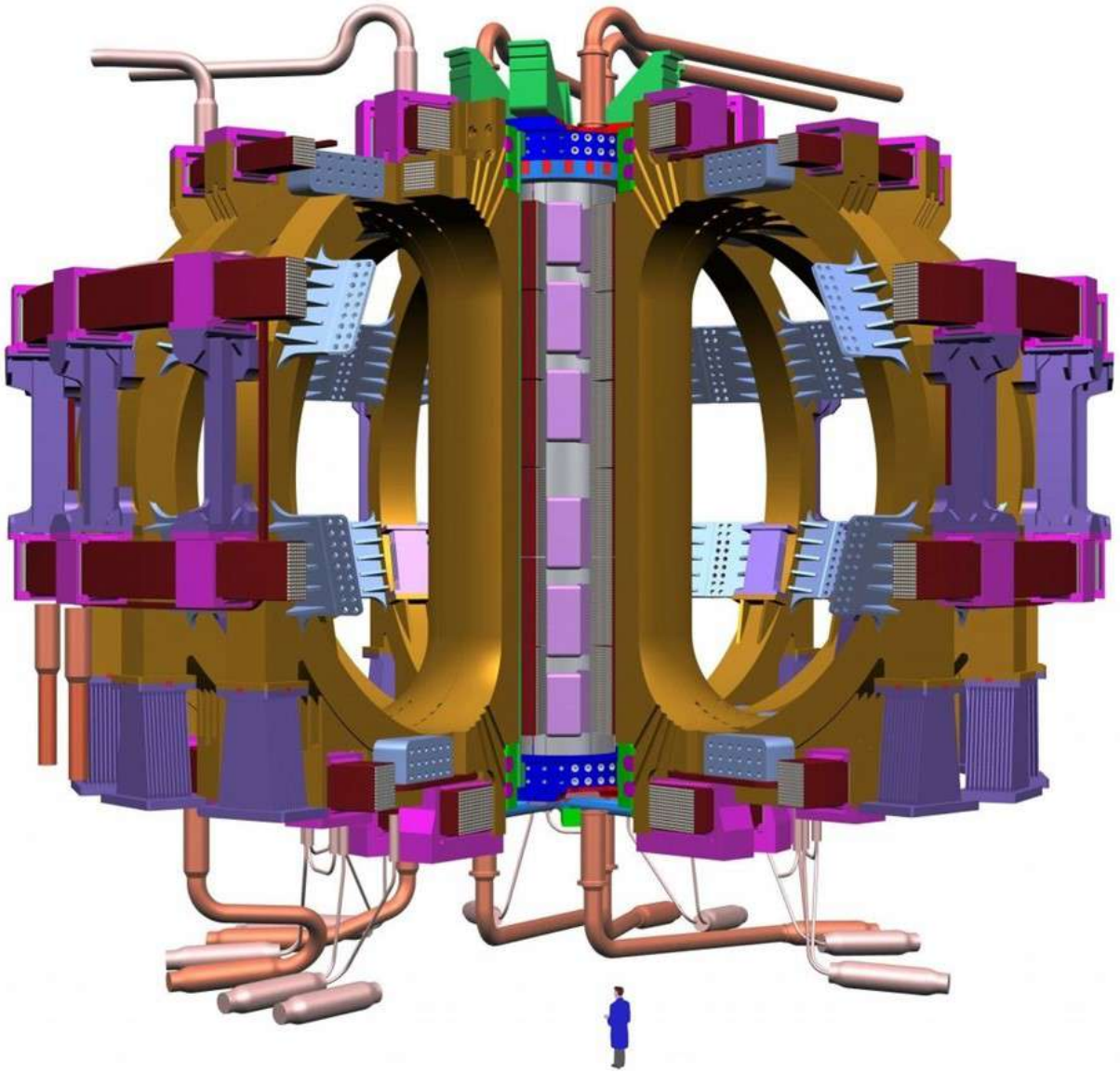


Source: iteryhiers.free.fr



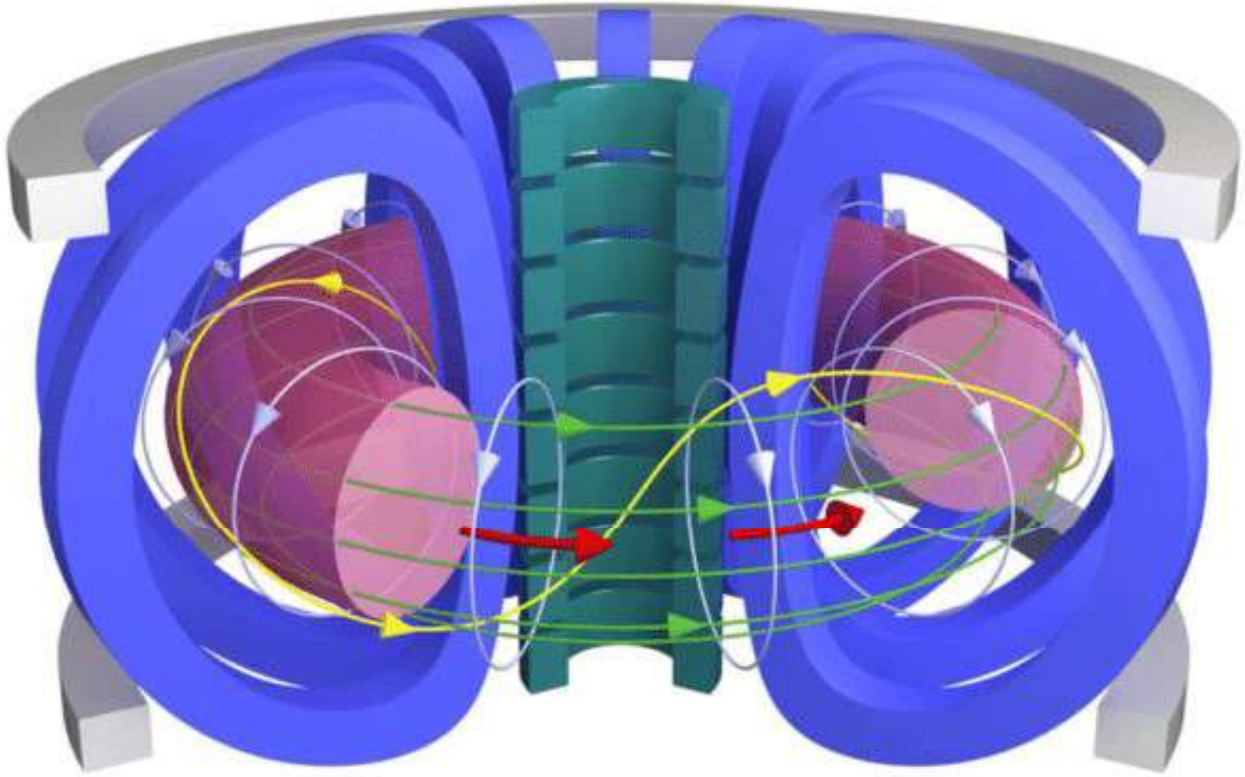
Source: iteryhiers.free.fr

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



Source: iter.org

Représentation Schématique du Tokamak ITER



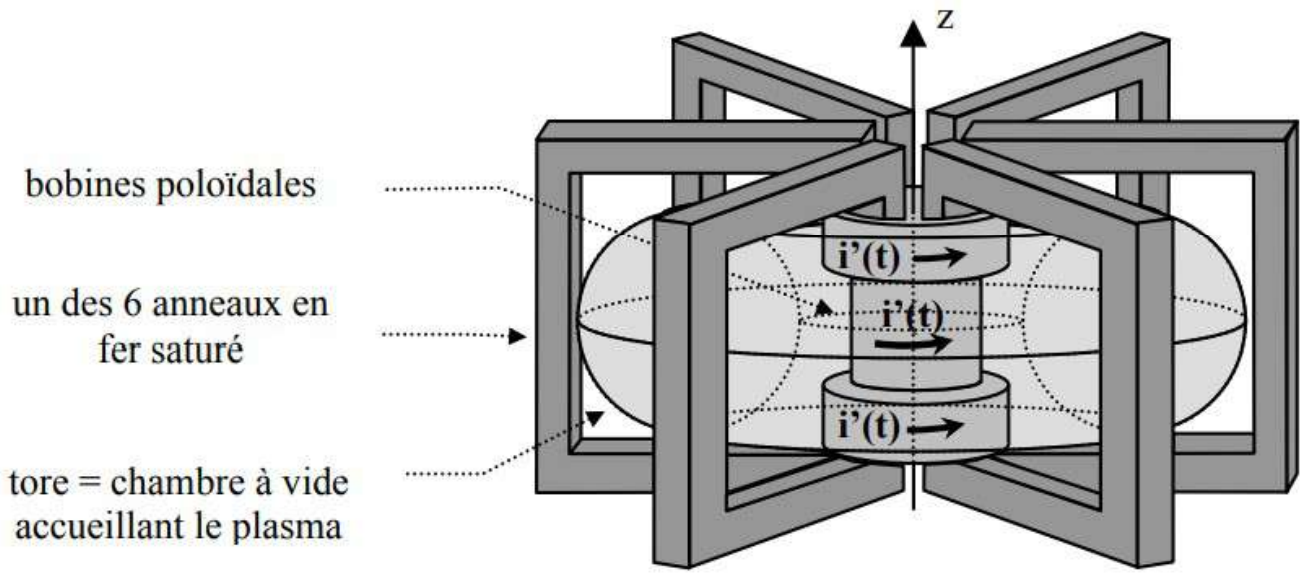
Bobines de champ toroïdales
Plasma confiné
Bobines internes de champ poloidal
Bobines externes de champ poloidal

Champ magnétique poloïdal
Champ magnétique toroïdal
Champ magnétique résultant
Courant plasma

Source: researchgate.net

Une seconde solution:
Ajout d'anneaux en fer saturé

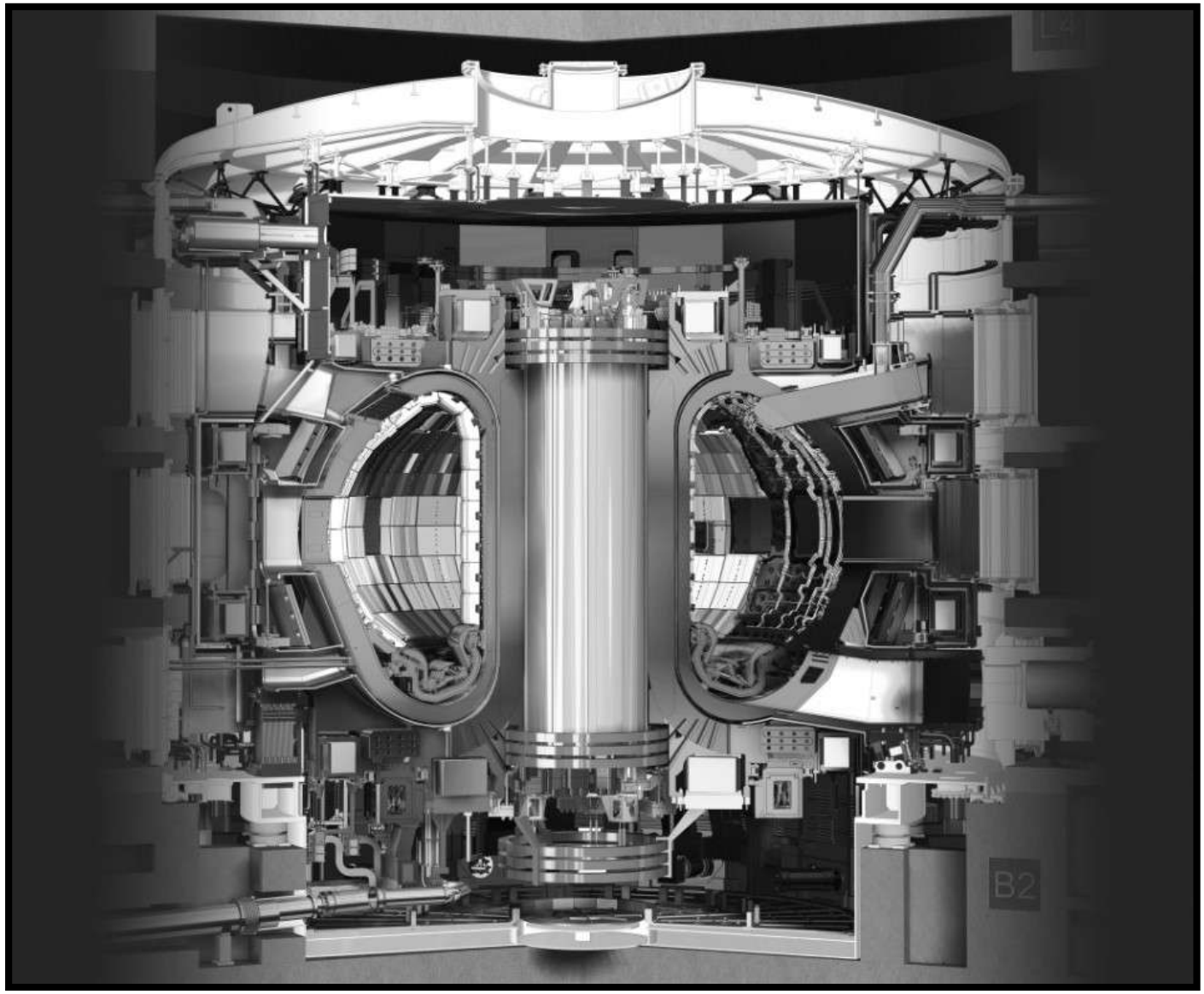
3) Pour contrecarrer cette dérive des particules chargées lorsque seul agit $\vec{B}_T$, on utilise un jeu de bobines coaxiales au tore, appelées bobines poloïdales, composées d'un grand nombre de spires parcourues par un courant variable i'(t), de fréquence ν = 50 Hz,. On associe à ces bobines 6 anneaux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

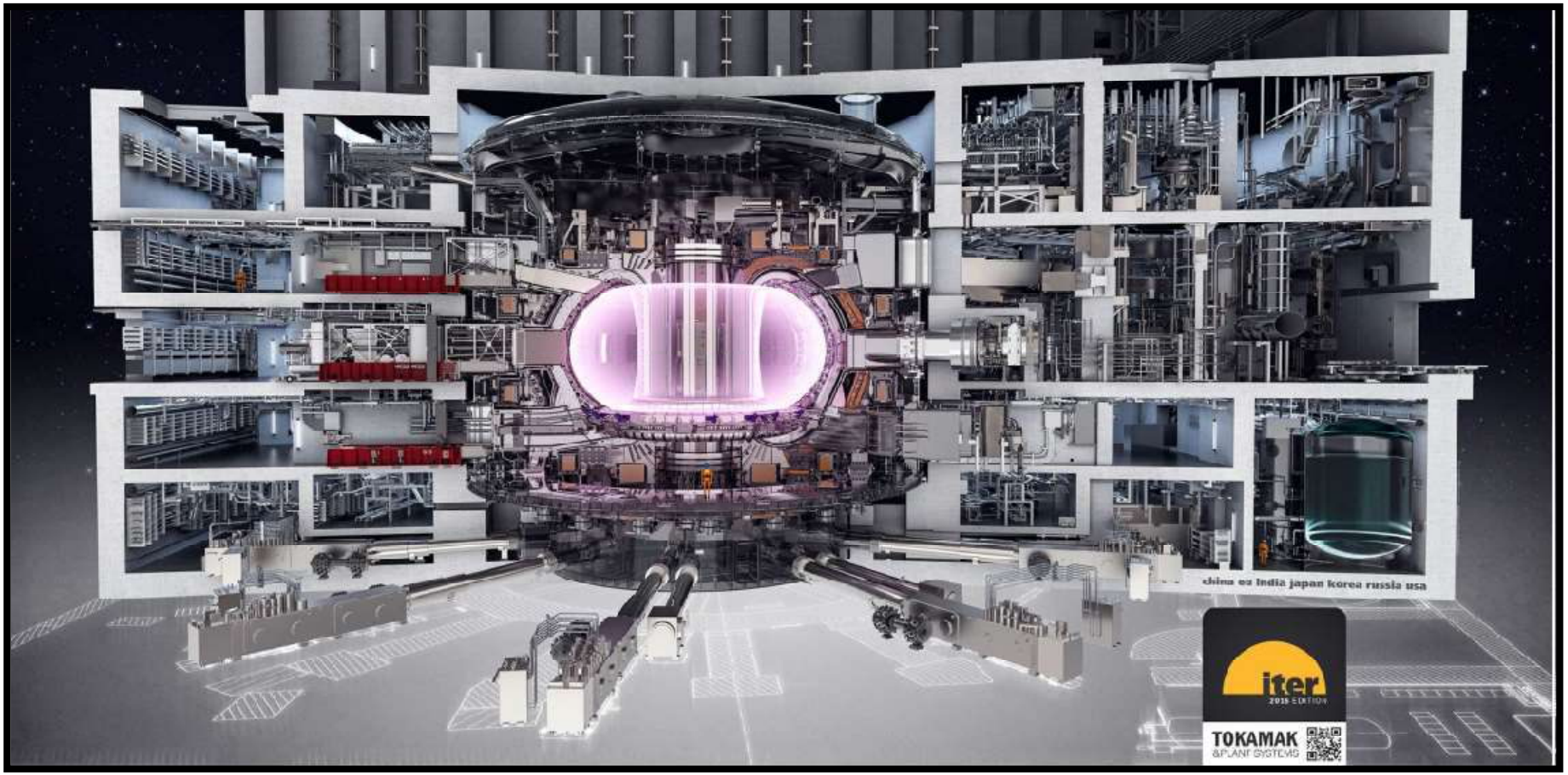
Source: ESIM 2003

Conclusion



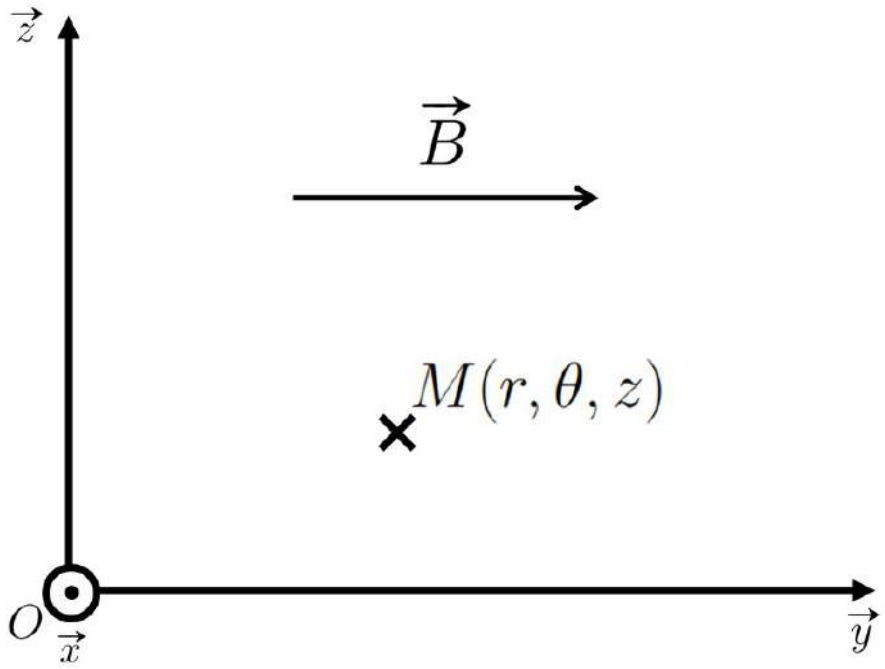
Source: iter.org

Annexes



Source: iter.org

Etude d'une particule chargée 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: $m\dfrac{d\vec{v}}{dt} = q\vec{v} \wedge \vec{B}$

$$m\frac{d\vec{v}}{dt} = \epsilon\omega_c \vec{v} \wedge \vec{u_y} \iff \begin{cases} \dot{v_x} = -\epsilon\omega_c v_z \\ \dot{v_y} = 0 \\ \dot{v_z} = \epsilon\omega_c v_x \end{cases}$$

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 $v_z(0) = 0$.

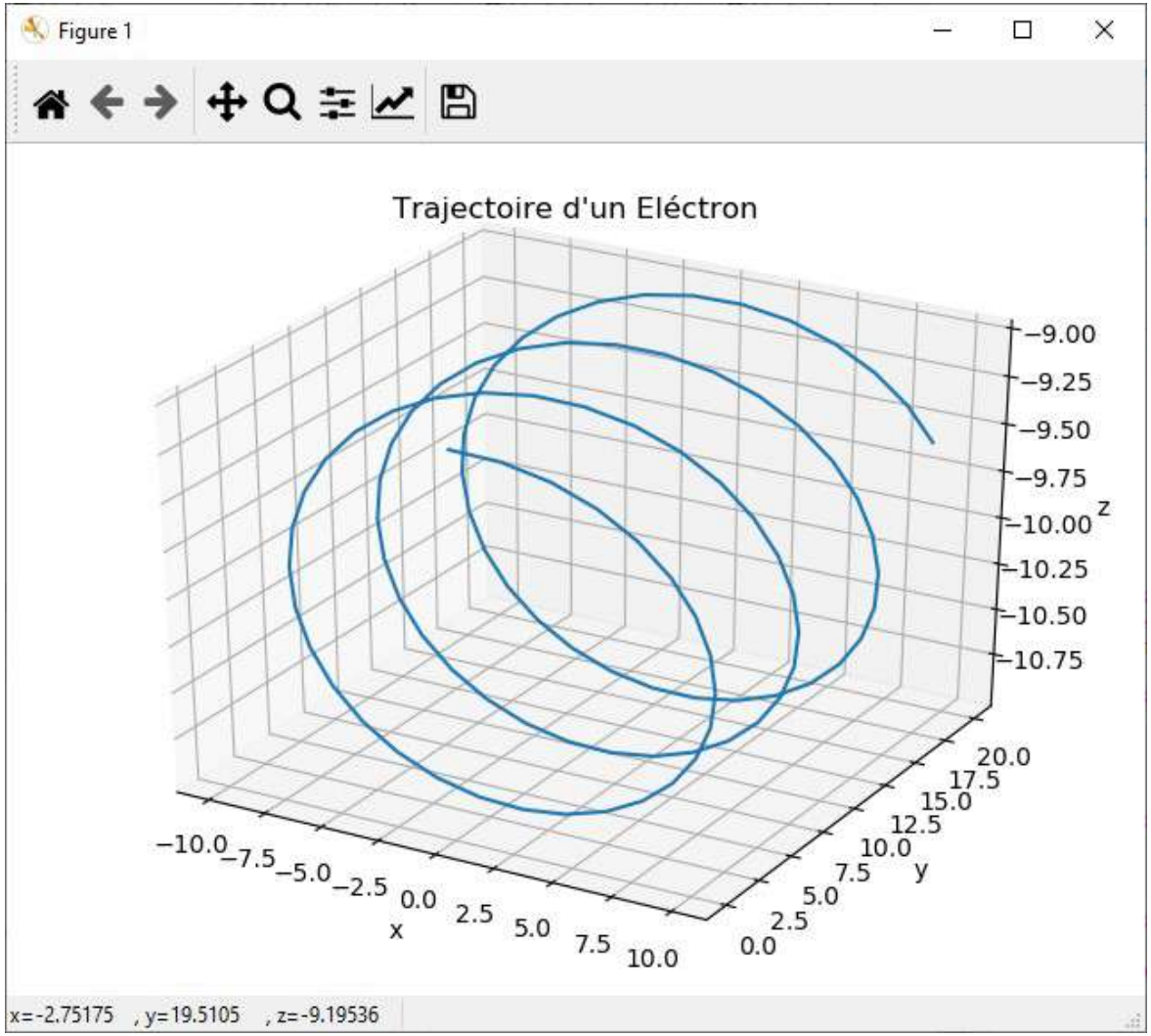
En découplant le système: $v_x + iv_z = v_\perp e^{i\epsilon\omega_c t}$

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

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

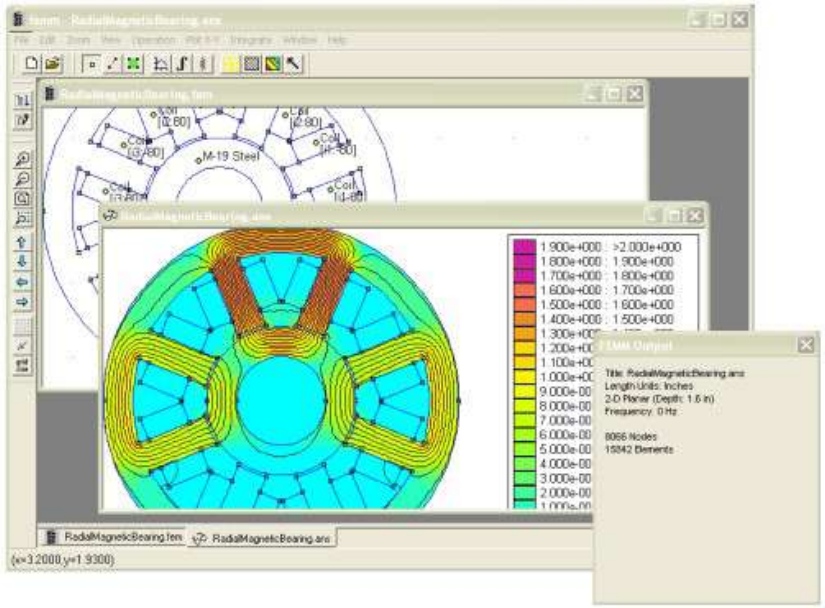


Logiciel FEMM: https://www.femm.info/wiki/HomePage

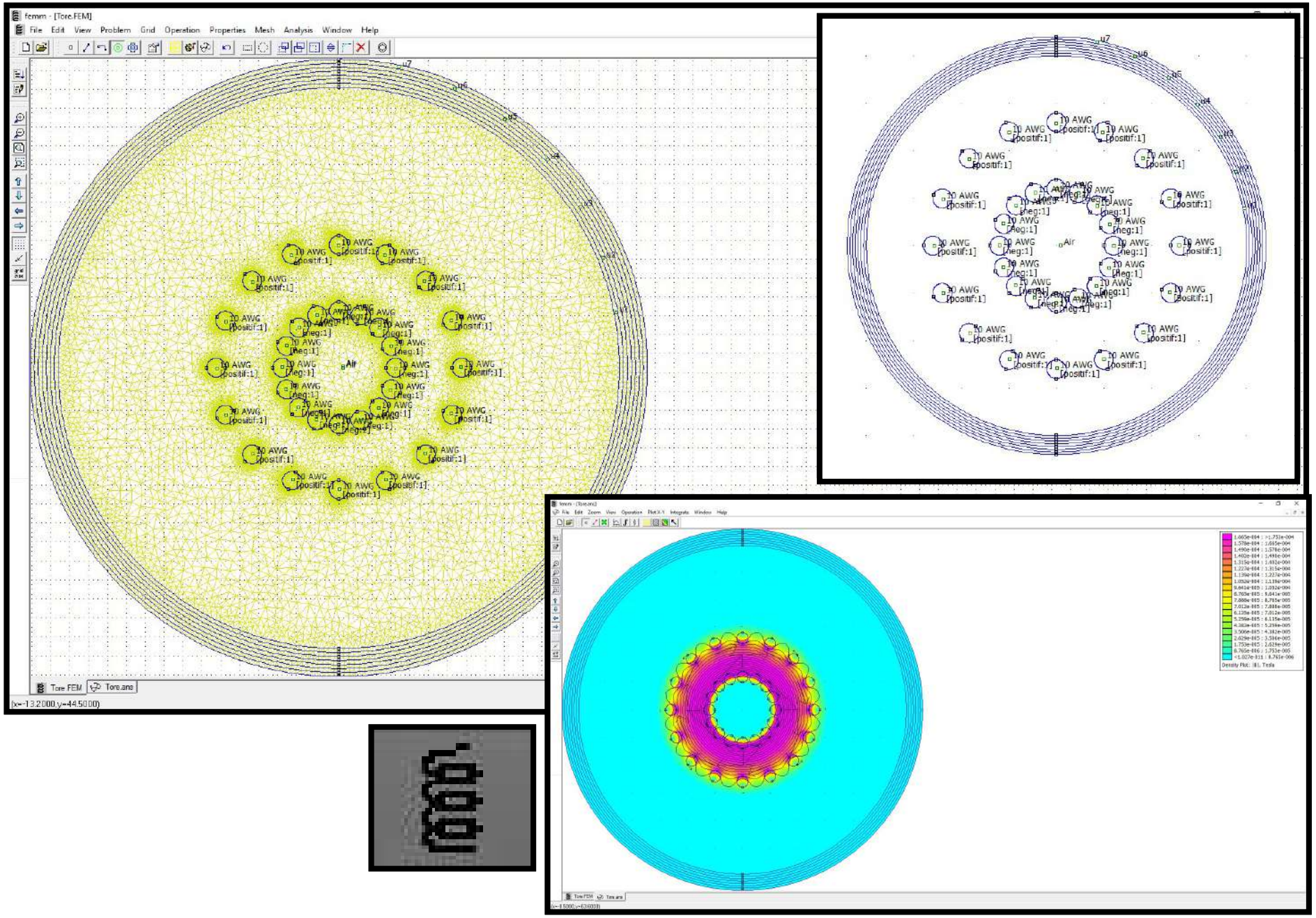
Finite Element Method Magnetics : HomePage

- Download
- Documentation
- FAQ
- Linux Support
- Examples
- User Contributions
- Miscellaneous
- Related Links
- Author

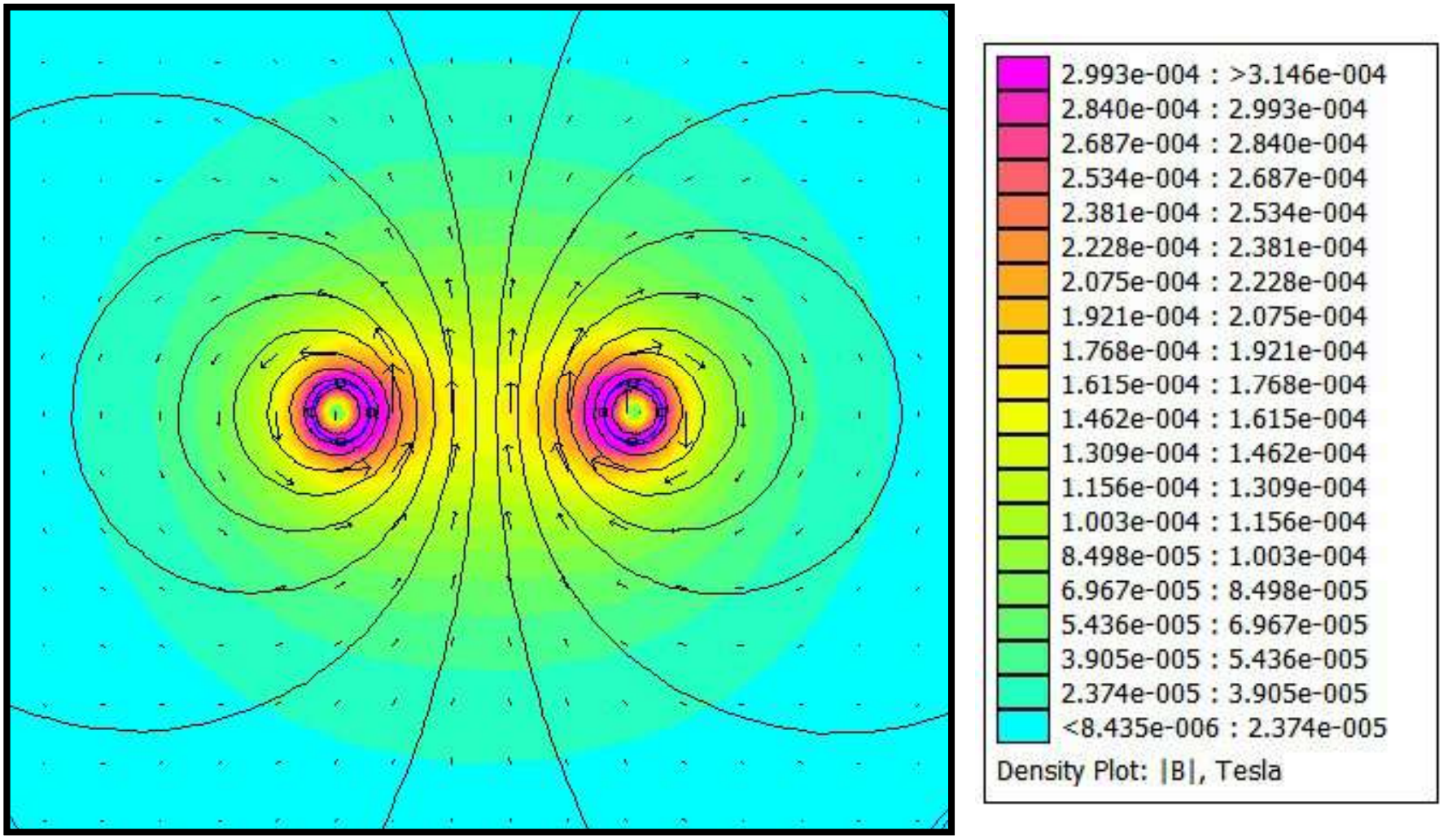
Finite Element Method Magnetics



Valid XHTML :: Valid CSS: :: Powered by WikkaWiki


femm - [Tore.FEM]
File Edit View Problem Grid Operation Properties Mesh Analysis Window Help
10 AWG [positif:1]
10 AWG [negif:1]
Air
Tore.FEM
Tore.ans

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

$${}^2_1D + {}^3_1T \rightarrow {}^4_2He + n \quad \text{avec l'énergie libérée} \quad E_f = +17,6 \text{ MeV}$$

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: $v_0 = 1,3.10^4 \text{ m/s}$

Choix de la vitesse initiale:

Thermodynamique statistique

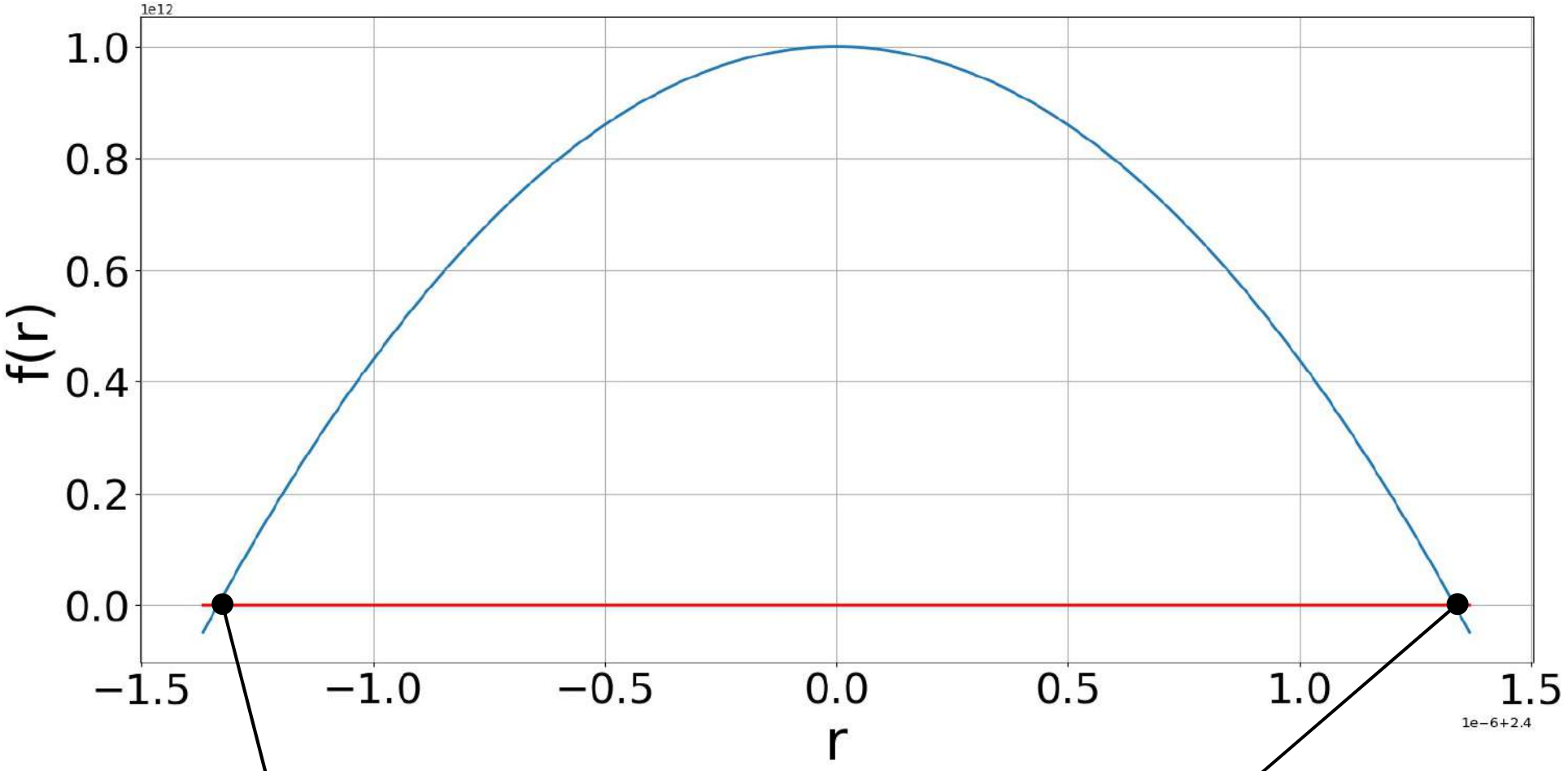
$$\langle \frac{1}{2} m v^2 \rangle = \frac{3 k_B T}{2} \iff \langle v \rangle = \sqrt{\frac{3 k_B T}{m}}$$

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

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

$$T = 150.10^6\,\mathrm{K} \Rightarrow v = 8\,.\,10^7\,\mathrm{m/s}$$

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



$R_1 = 2.399999$

$R_2 = 2.400001$

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

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

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

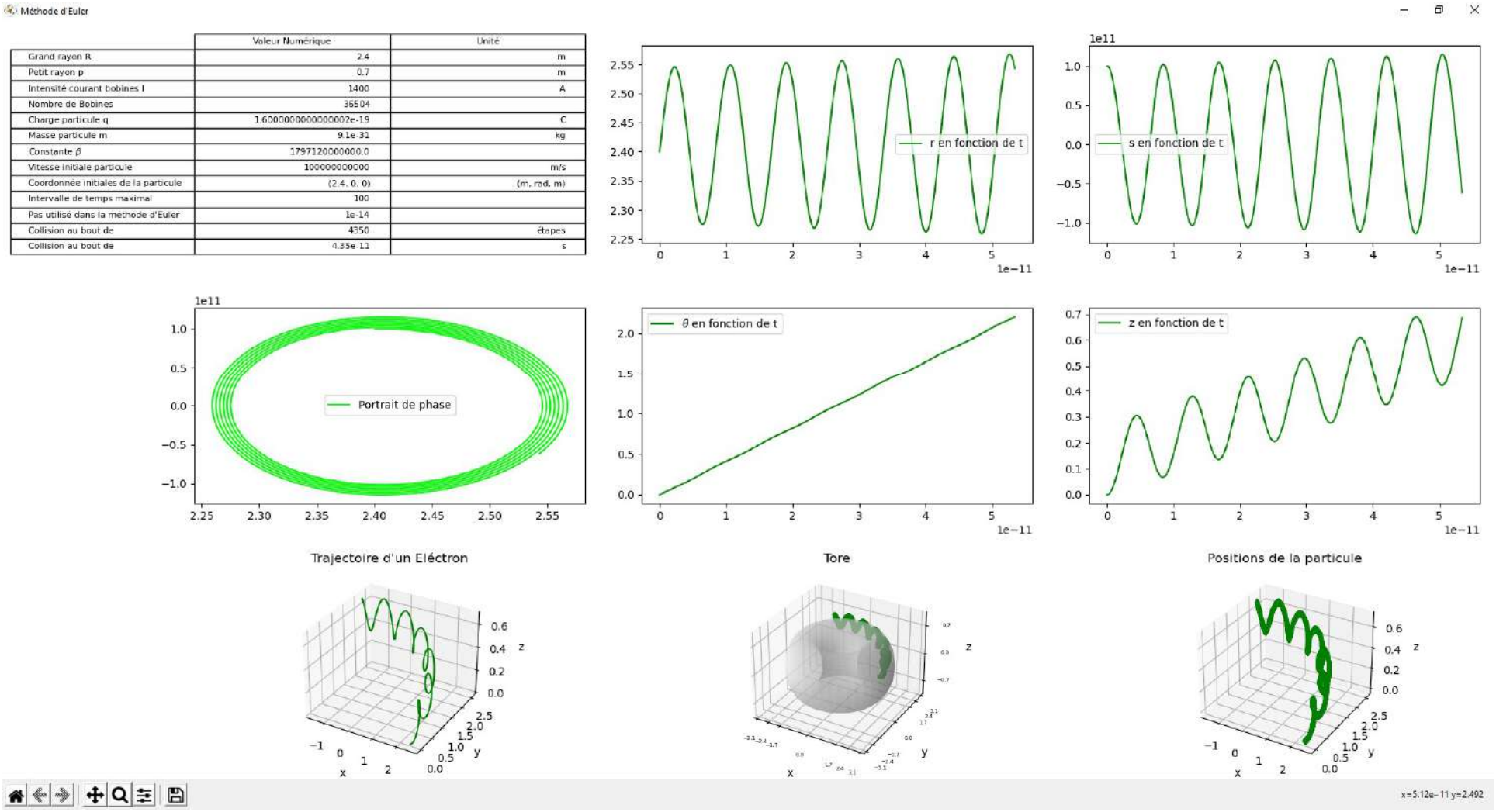
Discrétisation du temps: $Y_{n+1} = Y_n + \tau\, g(Y_n)$

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

$$\begin{cases} r_0 = R \\ s_0 = v_0 \\ \theta_0 = 0 \\ {}_0 = 0 \end{cases}$$

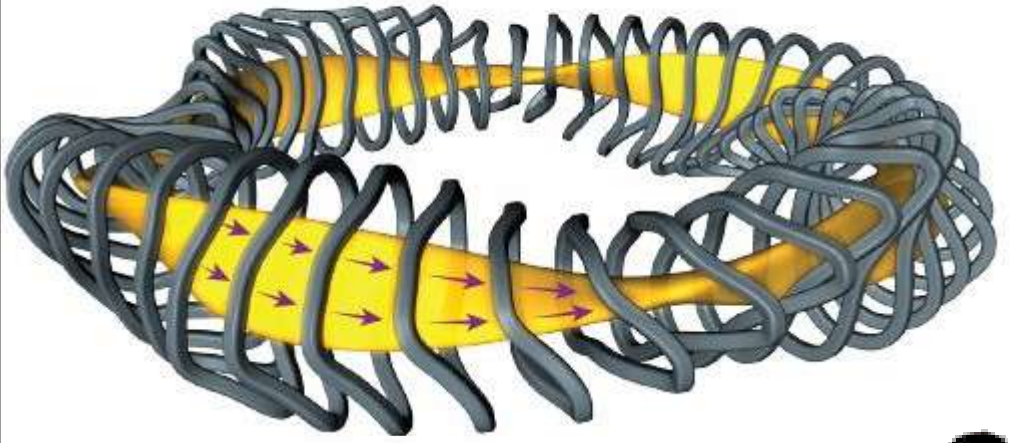
$$\begin{cases} r_{n+1} = r_n + \tau\, s_n \\ s_{n+1} = s_n + \tau \left(\dfrac{(Rv_0)^2}{r^3} - \dfrac{\beta^2}{r} \ln \dfrac{r}{R} \right) \\ \theta_{n+1} = \theta_n + \tau \dfrac{Rv_0}{r^2} \\ z_{n+1} = z_n + \tau\, \beta \ln \dfrac{r}{R} \end{cases}$$

Résultats Obtenus avec la Méthode d’Euler

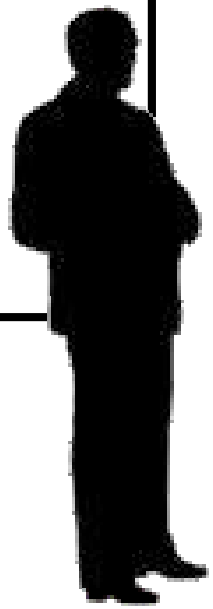


Autres systèmes de confinement magnétique:

Stellarator

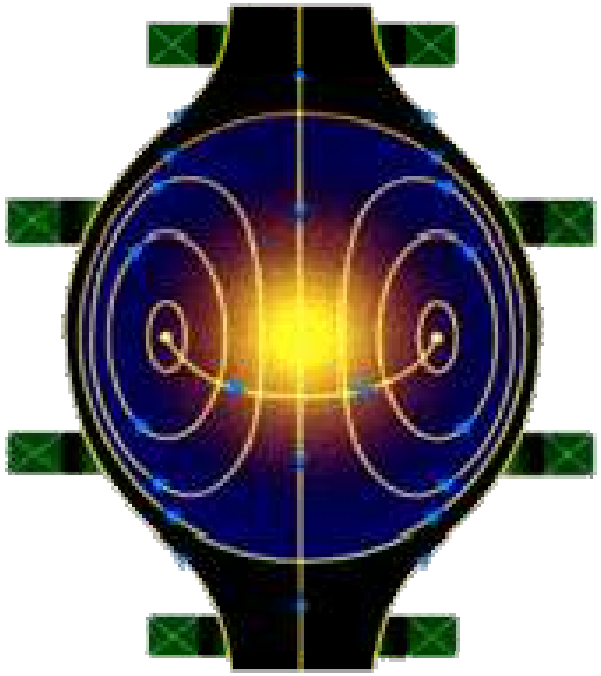


A complex asymmetric magnetic field ensures every plasma particle feels the same force, so the plasma can be maintained – CONTINUOUS OPERATION



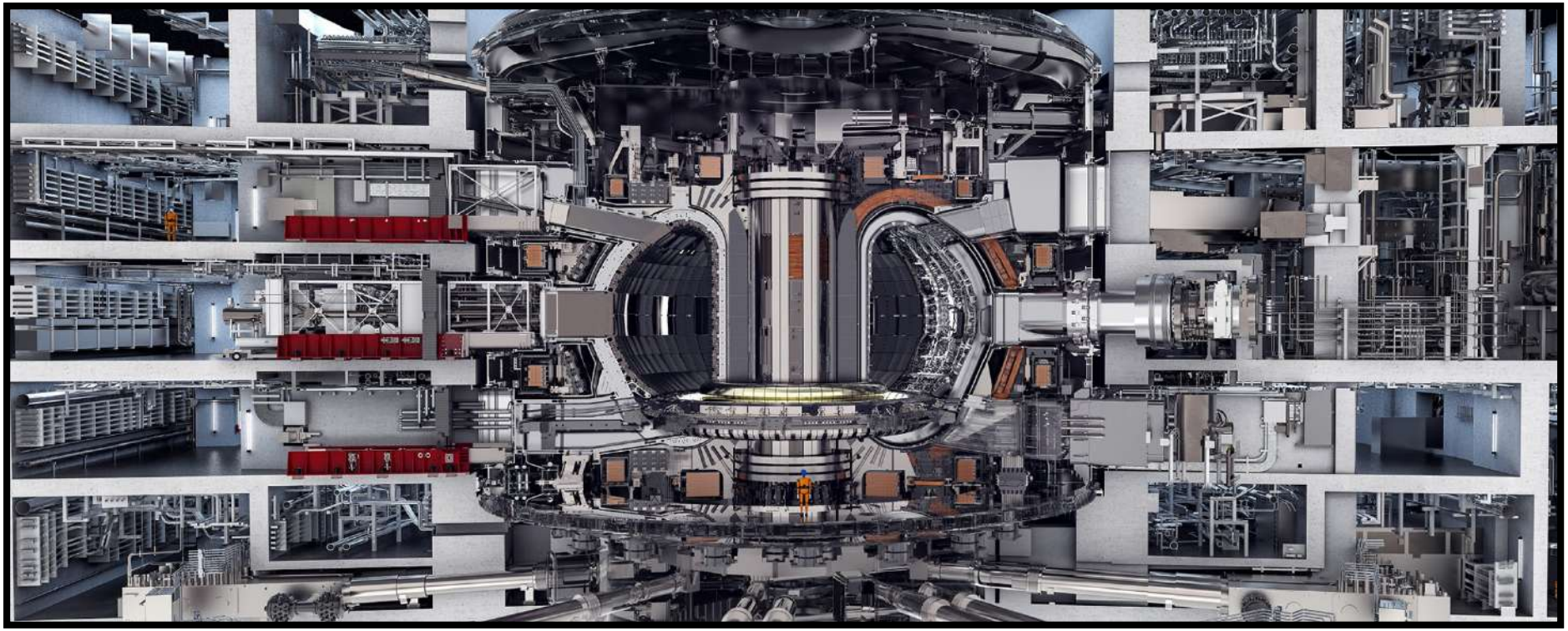
Source: @newscientist - twitter.com

Spheromak



Source: iterru.ru

Programmes Python



Source: iter.org

```
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| µ=4*pi*10**-7 #Perméabilité du vide
033| m=9.1*10**-31 #Masse e-
034|
035| B=(q*µ*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
s'annule avec la précision p
068|     while r2-r1>p:
069|         m=(r1+r2)*0.5
070|         if f(m)*f(r1)<0:
071|             r2=m
072|         else:
073|             r1=m
074|     return((r1+r2)*0.5)
075|
076| def valeurs_negatives(f,R):
077|     #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
078|     x1,x2=R,R
079|     while f(x1)>0:
080|         x1=x1/2
081|     while f(x2)>0:
082|         x2=2*x2
083|     x1=x1/2
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|     x1i,x2i=valeurs_negatives(f,R)
090|     x1=dichotomie(f,x1i,R,p)
091|     x2=dichotomie(f,R,x2i,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",r"Constante $\beta$","Vitesse initiale de la particule","Coordonnées
initiales de la particule","","Intervalle d'étude","R1 (premier point d'annulation de f)","R2 (deuxièmle 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):
```

```
    #converti coordonnées cylindriques en coordonnées cartésiennes
    x=r*cos(theta)
    y=r*sin(theta)
    return(x,y) # Faible précision

def mult_l_pol_cart(R_list,THETA,Z,T):
    #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)
    X,Y=[],[]
    for i in range(len(T)):
        x,y=mult_pol_cart(R_list[i],THETA[i])
        X.append(x)
        Y.append(y)
    return(X,Y,Z)

## Méthode d'Euler:

R_Euler,S_Euler,THETA_Euler,Z_Euler,T_Euler=[r0],[v0],[theta0],[z0],[t0] #Initialisation des listes de valeur

r,s,theta,z,t=r0,v0,theta0,z0,t0 #Initialisation des variables

def f_s(r):
    return (((R*v0)**2)/r**3)-(B**2)*(log(r/R)/r)

def f_theta(r):
    return (R*v0)/(r**2)

def f_z(r):
    return B*log(r/R)

i=0

while (t<=Tmax) and (est_dans_tore(r,z,rho,R)):
    print(est_dans_tore(r,z,rho,R))
    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
    i+=1
    print(color.BOLD + str(i) + color.END,"\n",r,s,theta,z,t,"\n")
    R_Euler.append(r)
    S_Euler.append(s)
    THETA_Euler.append(theta)
    Z_Euler.append(z)
    T_Euler.append(t)

## Visualisation des résultats obtenus avec la méthode d'Euler:

## r(t):
plt.figure()
plt.plot(T_Euler,R_Euler,color="green")
plt.xlabel("t",fontsize=30)
plt.ylabel("r(t)",fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.tight_layout()
plt.show()

## z(t):
plt.figure()
plt.plot(T_Euler,Z_Euler,color="green")
plt.xlabel("t",fontsize=30)
plt.ylabel("z(t)",fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.tight_layout()
plt.show()

## Général:
X,Y,Z=mult_l_pol_cart(R_Euler,THETA_Euler,Z_Euler,T_Euler)

euler1=plt.figure("Méthode d'Euler ",figsize = (16, 9))
plt.gcf().subplots_adjust(left = 0.1, bottom = 0.1,right = 0.98, top = 0.98, wspace = 0.4, hspace = 0.4)

ax = euler1.add_subplot(3, 3, 1) #Table
data=[[R,"m"],
      [rho,"m"],
      [I,"A"],
      [N,""],
      [q,"C"],
      [m,"kg"],
      [B,""],
      [v0,"m/s"],
      [(r0,theta0,z0),"(m, rad, m)"],
      [Tmax,""],
      [h,""],
      [i,"étapes"],
      [i*h,"s"]
      ]
```

```
column_labels=["Valeur Numérique","Unité"]
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"]
plt.axis('tight')
plt.axis('off')
plt.table(cellText=data,colLabels=column_labels,rowLabels=row_labels,loc="center")

ax = euler1.add_subplot(3, 3, 2)
plt.plot(T_Euler,R_Euler,label="r en fonction de t",color="green")
plt.legend()

ax = euler1.add_subplot(3, 3, 3)
plt.plot(T_Euler,S_Euler,label="s en fonction de t",color="green")
plt.legend()

ax = euler1.add_subplot(3, 3, 4) #Portrait de phase
plt.plot(R_Euler,S_Euler,label="Portrait de phase",color="lime")
plt.legend()

ax = euler1.add_subplot(3, 3, 5)
plt.plot(T_Euler,THETA_Euler,label=r"$\theta$ en fonction de t",color="green")
plt.legend()

ax=euler1.add_subplot(3,3,6)
plt.plot(T_Euler,Z_Euler,label="z en fonction de t",color="green")
plt.legend()

ax=euler1.add_subplot(3,3,7, projection='3d') #Trajectoire

ax.plot(X, Y, Z, label='Courbe',color="green")
plt.title("Trajectoire d'un Eléctron")
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.tight_layout()

ax=euler1.add_subplot(3,3,8, projection='3d') #Tore
(theta, phi) = np.meshgrid(np.linspace(0, 2 * np.pi, 41),
                           np.linspace(0, 2 * np.pi, 41))

x = (1.9 + np.cos(phi)) * np.cos(theta)
y = (1.9 + np.cos(phi)) * np.sin(theta)
z = 0.7*np.sin(phi)

dplot = ax.plot_surface(x,y,z,color='whitesmoke' ,alpha=0.5)
ax.set(xlabel='x',
       ylabel='y',
       zlabel='z',
       xlim = [-3.2, 3.2],
       ylim = [-3.2, 3.2],
       zlim = [-1, 1],
       xticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
       yticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
       zticks = [-0.7, 0, 0.7],
       title='Tore')
for t in ax.xaxis.get_major_ticks(): t.label.set_fontsize(5)
for t in ax.yaxis.get_major_ticks(): t.label.set_fontsize(5)
for t in ax.zaxis.get_major_ticks(): t.label.set_fontsize(5)
ax.scatter(X, Y, Z, marker='p',color="green")  # Tracé des points 3D

ax=euler1.add_subplot(3,3,9, projection='3d')
ax.scatter(X, Y, Z, marker='d',color="green")  # Tracé des points 3D
plt.title("Positions de la particule")
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.tight_layout()

plt.show()

## Trajectoire:
traj = plt.figure()
X,Y,Z=mult_l_pol_cart(R_list,THETA,Z,T)
ax = traj.gca(projection='3d')  # Affichage en 3D
ax.plot(X, Y, Z, label='Courbe')  # Tracé de la courbe 3D
plt.title("Trajectoire d'un Eléctron")
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.tight_layout()
plt.show()

## Tore:
X,Y,Z=mult_l_pol_cart(R_Euler,THETA_Euler,Z_Euler,T_Euler)
tore = plt.figure("Tore",clear=True)
```

```
ax = tore.add_subplot(1, 1, 1, projection='3d')

(theta, phi) = np.meshgrid(np.linspace(0, 2 * np.pi, 41),
                           np.linspace(0, 2 * np.pi, 41))

x = (1.9 + np.cos(phi)) * np.cos(theta)
y = (1.9 + np.cos(phi)) * np.sin(theta)
z = 0.7*np.sin(phi)

dplot = ax.plot_surface(x,y,z,color='whitesmoke' ,alpha=0.5)
ax.set(xlabel='x',
       ylabel='y',
       zlabel='z',
       xlim = [-3.2, 3.2],
       ylim = [-3.2, 3.2],
       zlim = [-1, 1],
       xticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
       yticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
       zticks = [-0.7, 0, 0.7],
       title='Tore')
ax.scatter(X, Y, Z, marker='p')
tore.tight_layout()
plt.show()
##
for angle in range(0, 360):
    ax.view_init(angle, angle)
    plt.draw()
    plt.pause(.001)

## Méthode de Heun:

def f_s(r):
    return (((R*v0)**2)/r**3)-(B**2)*(log(r/R)/r)

def f_theta(r):
    return (R*v0)/(r**2)

def f_z(r):
    return B*log(r/R)

r1,s1,theta1,z1=r0,v0,theta0,z0 #Initialisation des variables

r,s,theta,z,t=r0,v0,theta0,z0,t0 #Initialisation des variables

R_Heun,S_Heun,THETA_Heun,Z_Heun,T_Heun=[r0],[v0],[theta0],[z0],[t0] #Initialisation des listes de valeur

i=0

while (t<=Tmax) and (est_dans_tore(r,z,rho,R)):
    #Méthode d'Euler pour approximation:
    print(est_dans_tore(r,z,rho,R))
    r1, s1, theta1, z1= r1+h*s1, s1+h*f_s(r1), theta1+h*f_theta(r1), z1+h*f_z(r1)

    #Méthode de Heun:
    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,
z+h*(f_z(r)+f_z(r1))/2, t+h
    i+=1
    print(i,"\n",r,s,theta,z,t,"\n")
    R_Heun.append(r)
    S_Heun.append(s)
    THETA_Heun.append(theta)
    Z_Heun.append(z)
    T_Heun.append(t)

## Visualisation des résultats obtenus avec la méthode de Heun:

## r(t):
plt.figure()
plt.plot(T_Heun,R_Heun,color="darkorange")
plt.xlabel("t",fontsize=30)
plt.ylabel("r(t)",fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.tight_layout()
plt.show()

## z(t):
plt.figure()
plt.plot(T_Heun,Z_Heun,color="darkorange")
plt.xlabel("t",fontsize=30)
plt.ylabel("z(t)",fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.tight_layout()
plt.show()

## Comparasion Euler/Heun -> r(t):
```

```
plt.figure()
plt.plot(T_Euler,R_Euler,label="Euler",color="green")
plt.plot(T_Heun,R_Heun,"--",label="Heun",color="darkorange")
plt.xlabel("t",fontsize=30)
plt.ylabel("r(t)",fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.legend(loc="right")
plt.tight_layout()
plt.show()

## Comparasion Euler/Heun -> z(t):
plt.figure()
plt.plot(T_Euler,Z_Euler,label="Euler",color="green")
plt.plot(T_Heun,Z_Heun,"--",label="Heun",color="darkorange")
plt.xlabel("t",fontsize=30)
plt.ylabel("z(t)",fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.tight_layout()
plt.legend(loc="right")
plt.show()

## Comparasion Euler/Heun -> r(t) (zoom):
i=4350
j=4450
echantillon_Euler=R_Euler[i:j]
echantillon_Euler_T=T_Euler[i:j]
echantillon_Heun=R_Heun[i:j]
echantillon_Heun_T=T_Heun[i:j]

plt.figure()
plt.plot(echantillon_Euler_T,echantillon_Euler,label="Euler",color="green")
plt.plot(echantillon_Heun_T,echantillon_Heun,label="Heun",color="darkorange")
plt.xlabel("t",fontsize=30)
plt.ylabel("r(t)",fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.legend(loc="right")
plt.tight_layout()
plt.show()

## Général:
X,Y,Z=mult_l_pol_cart(R_Heun,THETA_Heun,Z_Heun,T_Heun)

heun=plt.figure("Méthode de Heun",figsize = (16, 9))
#plt.title("Méthode d'Euler 1")
plt.gcf().subplots_adjust(left = 0.1, bottom = 0.1,right = 0.98, top = 0.98, wspace = 0.4, hspace = 0.4)


ax = heun.add_subplot(3, 3, 1) #Table
data=[[R,"m"],
      [rho,"m"],
      [I,"A"],
      [N,""],
      [q,"C"],
      [m,"kg"],
      [B,""],
      [v0,"m/s"],
      [(r0,theta0,z0),"(m, rad, m)"],
      [Tmax,""],
      [h,""],
      [i,"étapes"],
      [i*h,"s"]
      ]
column_labels=["Valeur Numérique","Unité"]
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 maximal","Pas utilisé dans la méthode d'Euler","Collision au bout
de","Collision au bout de"]
plt.axis('tight')
plt.axis('off')
plt.table(cellText=data,colLabels=column_labels,rowLabels=row_labels,loc="center")

ax = heun.add_subplot(3, 3, 2)
plt.plot(T_Heun,R_Heun,label="r en fonction de t",color="darkorange")
plt.legend()

ax = heun.add_subplot(3, 3, 3)
plt.plot(T_Heun,S_Heun,label="s en fonction de t",color="darkorange")
plt.legend()

ax = heun.add_subplot(3, 3, 4) #Portrait de phase
plt.plot(R_Heun,S_Heun,label="Portrait de phase",color="yellow")
plt.legend()

ax = heun.add_subplot(3, 3, 5)
```

```
plt.plot(T_Heun,THETA_Heun,label=r"$\theta$ en fonction de t",color="darkorange")
plt.legend()

ax=heun.add_subplot(3,3,6)
plt.plot(T_Heun,Z_Heun,label="z en fonction de t",color="darkorange")
plt.legend()

ax=heun.add_subplot(3,3,7, projection='3d') #Trajectoire

ax.plot(X, Y, Z, label='Courbe',color="darkorange")
plt.title("Trajectoire d'un Eléctron")
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.tight_layout()

ax=heun.add_subplot(3,3,8, projection='3d') #Tore
(theta, phi) = np.meshgrid(np.linspace(0, 2 * np.pi, 41),
                           np.linspace(0, 2 * np.pi, 41))

x = (1.9 + np.cos(phi)) * np.cos(theta)
y = (1.9 + np.cos(phi)) * np.sin(theta)
z = 0.7*np.sin(phi)

dplot = ax.plot_surface(x,y,z,color='whitesmoke' ,alpha=0.5)
ax.set(xlabel='x',
       ylabel='y',
       zlabel='z',
       xlim = [-3.2, 3.2],
       ylim = [-3.2, 3.2],
       zlim = [-1, 1],
       xticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
       yticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
       zticks = [-0.7, 0, 0.7],
       title='Tore')
for t in ax.xaxis.get_major_ticks(): t.label.set_fontsize(5)
for t in ax.yaxis.get_major_ticks(): t.label.set_fontsize(5)
for t in ax.zaxis.get_major_ticks(): t.label.set_fontsize(5)
ax.scatter(X, Y, Z, marker='p',color="darkorange")  # Tracé des points 3D

ax=heun.add_subplot(3,3,9, projection='3d')
ax.scatter(X, Y, Z, marker='d',color="darkorange")  # Tracé des points 3D
plt.title("Positions de la particule")
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.tight_layout()

plt.show()

## Méthode utilisant le module Scipy:

def instant_collision(dans_tore):
    i=0
    intervalle_temp=T_calcul/nbr_points
    for i in range(len(dans_tore)):
        if dans_tore[i]<=0:
            return(i,i*intervalle_temp)
    return("pas de sortie de tore","pas de sortie de tore")


def f_s(r):
    return (((R*v0)**2)/r**3)-(B**2)*(log(r/R)/r)

def f_theta(r):
    return (R*v0)/(r**2)

def f_z(r):
    return B*log(r/R)

def dY(Y,t):
    r,s,theta,z=Y
    dYdt=[s,f_s(r),f_theta(r),f_z(r)]
    return dYdt

Y0=[r0,v0,theta0,z0] #CI

T_calcul=10**-10
nbr_points=10000
t=np.linspace(0,T_calcul,nbr_points)

from scipy.integrate import odeint
sol = odeint(dY,Y0,t)
#sol = odeint(dY,Y0,t,mxstep=50000000)

dans_tore=[rho**2 -(sol[i,0]-R)**2-sol[i,3]**2 for i in range(len(sol))]
Indice_collision,T_collision=instant_collision(dans_tore)
```

```
print(Indice_collision,T_collision)

## Visualisation des résultats obtenus avec la méthode Scipy:

## Général:
solg=plt.figure("Méthode scipy",figsize = (16, 9))
plt.gcf().subplots_adjust(left = 0.13, bottom = 0.1,right = 0.98, top = 0.95, wspace = 0.2, hspace = 0.4)

ax = solg.add_subplot(3, 3, 1) #Table
data=[[R,"m"],
      [rho,"m"],
      [I,"A"],
      [N,""],
      [q,"C"],
      [m,"kg"],
      [B,""],
      [v0,"m/s"],
      [(r0,theta0,z0),"(m, rad, m)"],
      [T_calcul,"s"],
      [T_collision,"s"]
      ]
column_labels=["Valeur Numérique","Unité"]
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"]
plt.axis('tight')
plt.axis('off')
plt.table(cellText=data,colLabels=column_labels,rowLabels=row_labels,loc="center")

ax = solg.add_subplot(3, 3, 2)
plt.plot(t,sol[:,0],label="r en fonction de t")
plt.legend()

ax = solg.add_subplot(3, 3, 3)
plt.plot(t,sol[:,1],label="s en fonction de t")
plt.legend()

ax = solg.add_subplot(3, 3, 4) #Portrait de phase
plt.plot(sol[:,0],sol[:,1],label="Portrait de phase",color="lightblue")
plt.legend()

ax = solg.add_subplot(3, 3, 5)
plt.plot(t,sol[:,2],label=r"$\theta$ en fonction de t")
plt.legend()

ax=solg.add_subplot(3,3,6)
plt.plot(t,sol[:,3],label="z en fonction de t")
plt.legend()

ax=solg.add_subplot(3,3,8)
plt.plot(t,dans_tore,label="positif<=>particule dans le tore")
plt.yticks([0])
plt.grid()

plt.show()

## r(t):
plt.figure()
plt.plot(t, sol[:, 0], label='r(t)')
plt.legend(loc='best')
plt.xlabel('t',fontsize=30)
plt.ylabel('r(t)',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.tight_layout()
plt.show()

## r(t) (avec horizontales):
x1p,x2p=zone_tore(f,R,p_dicho)
print(x1p,x2p)

plt.figure()
Y1=[x1p for i in range(len(sol[:, 0]))]
Y2=[x2p for i in range(len(sol[:, 0]))]
plt.plot(t, sol[:, 0], label='r(t)')
plt.plot(t,Y1,"red")
plt.plot(t,Y2,"red")
plt.legend(loc='best')
plt.xlabel('t',fontsize=30)
plt.ylabel('r(t)',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.tight_layout()
plt.show()

## z(t):
plt.figure()
```

```
plt.plot(t, sol[:, 3], label='z(t)')
plt.legend(loc='best')
plt.xlabel('t',fontsize=30)
plt.ylabel('z(t)',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.tight_layout()
plt.show()

## Comparasion méthode Euler/Scipy:
plt.figure()
plt.plot(t, sol[:, 0], label='Scipy',linewidth=2.2)
plt.plot(T_Euler,R_Euler,label="Euler",linewidth=1, color="green")
plt.legend(loc='best')
plt.xlim(0,0.6*10**-10)
plt.xlabel('t',fontsize=30)
plt.ylabel('r(t)',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.tight_layout()
plt.show()

## Négatif <=> Sortie du tore:

plt.figure()
tore=[rho**2 -(sol[i,0]-R)**2-sol[i,3]**2 for i in range(len(sol))]

plt.plot(t,tore)
plt.xticks(fontsize=20)
plt.yticks([0],fontsize=20)
plt.grid()
plt.tight_layout()
plt.show()

##Négatif <=> Sortie du tore (réduit):
plt.figure()
plt.plot(t,dans_tore,label="positif<=>particule dans le tore")
plt.plot(t,[0 for i in range(len(t))],color="red")
plt.plot([T_collision for i in range(10)],[i-1 for i in range(10)],color="red")
plt.yticks([0],fontsize=20)
plt.xticks([0,5.4*10**-11,6*10**-11],fontsize=20)
#plt.xticks(fontsize=20)
plt.xlim(0,6*10**-11)
plt.ylim(-0.1,0.5)
plt.grid()
plt.tight_layout()
plt.show()

##Tore:

def pol_cart(list):
    r,theta,z = list[0],list[2],list[3]
    x=r*cos(theta)
    y=r*sin(theta)
    return(x,y,z)

def l_pol_cart(sol):
    X,Y,Z=[],[],[]
    for i in range(len(sol)):
        x,y,z=pol_cart(sol[i])
        X.append(x)
        Y.append(y)
        Z.append(z)
    return(X,Y,Z)

X,Y,Z=l_pol_cart(sol[:540])
tore = plt.figure("Tore",clear=True)
ax = tore.add_subplot(1, 1, 1, projection='3d')

(theta, phi) = np.meshgrid(np.linspace(0, 2 * np.pi, 41),
                           np.linspace(0, 2 * np.pi, 41))

x = (1.9 + np.cos(phi)) * np.cos(theta)
y = (1.9 + np.cos(phi)) * np.sin(theta)
z = 0.7*np.sin(phi)

dplot = ax.plot_surface(x,y,z,color='whitesmoke' ,alpha=0.5)
ax.set(xlabel='x',
       ylabel='y',
       zlabel='z',
       xlim = [-3.2, 3.2],
       ylim = [-3.2, 3.2],
       zlim = [-1, 1],
       xticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
       yticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
       zticks = [-0.7, 0, 0.7],
       title='Tore')
```

```
767| ax.scatter(X, Y, Z, marker='p')
768| tore.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| µ=4*pi*10**-7 #Perméabilité du vide
017| m=9.1*10**-31 #Masse e-
018| B=(q*µ*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
```

```
    x1i,x2i=valeurs_negatives(f,R)
    x1=dichotomie(f,x1i,R,p)
    x2=dichotomie(f,R,x2i,p)
    return(x1,x2)

def instant_collision(dans_tore):
    i=0
    intervalle_temp=T_calcul/nbr_points
    for i in range(len(dans_tore)):
        if dans_tore[i]<=0:
            return(i,i*intervalle_temp)
    return("pas de sortie de tore","pas de sortie de tore")

## Enregistrement:

def f_s(r):
    return (((R*v0)**2)/r**3)-(B**2)*(log(r/R)/r)

def f_theta(r):
    return (R*v0)/(r**2)

def f_z(r):
    return B*log(r/R)

def dY(Y,t):
    r,s,theta,z=Y
    dYdt=[s,f_s(r),f_theta(r),f_z(r)]
    return dYdt

Y0=[r0,v0,theta0,z0] #CI

from scipy.integrate import odeint
sol = odeint(dY,Y0,t,mxstep=maxstep)
#sol = odeint(dY,Y0,t)

dans_tore=[rho**2 -(sol[i,0]-R)**2-sol[i,3]**2 for i in range(len(sol))]
Indice_collision,T_collision=instant_collision(dans_tore)
print(Indice_collision,T_collision)

file = open(Racine+FileName+".csv","w")
FileWriteCsv(file,sol)
file.close()

## Lecture:

#file = open(Racine+FileName+".csv","r")
file = open(Racine+"10000000000_9.000000000000001e-09_10000_1000000.csv","r")
sol = FileReadCsv(file)
file.close()

dans_tore=[rho**2 -(sol[i,0]-R)**2-sol[i,3]**2 for i in range(len(sol))]
Indice_collision,T_collision=instant_collision(dans_tore)
print(Indice_collision,T_collision)

## Visualisation:

## Générale:
solg=plt.figure("Méthode scipy",figsize = (16, 9))
plt.gcf().subplots_adjust(left = 0.13, bottom = 0.1,right = 0.98, top = 0.95, wspace = 0.2, hspace = 0.4)

ax = solg.add_subplot(3, 3, 1) #Table
data=[[R,"m"],
      [rho,"m"],
      [I,"A"],
      [N,""],
      [q,"C"],
      [m,"kg"],
      [B,""],
      [v0,"m/s"],
      [(r0,theta0,z0),"(m, rad, m)"],
      [T_calcul,"s"],
      [T_collision,"s"]
      ]
column_labels=["Valeur Numérique","Unité"]
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"]
plt.axis('tight')
plt.axis('off')
plt.table(cellText=data,colLabels=column_labels,rowLabels=row_labels,loc="center")

ax = solg.add_subplot(3, 3, 2)
plt.plot(t,sol[:,0],label="r en fonction de t")
plt.legend()

ax = solg.add_subplot(3, 3, 3)
plt.plot(t,sol[:,1],label="s en fonction de t")
```

```
plt.legend()

ax = solg.add_subplot(3, 3, 4) #Portrait de phase
plt.plot(sol[:,0],sol[:,1],label="Portrait de phase",color="lightblue")
plt.legend()

ax = solg.add_subplot(3, 3, 5)
plt.plot(t,sol[:,2],label=r"$\theta$ en fonction de t")
plt.legend()

ax=solg.add_subplot(3,3,6)
plt.plot(t,sol[:,3],label="z en fonction de t")
plt.legend()

ax=solg.add_subplot(3,3,8)
plt.plot(t,dans_tore,label="positif<=>particule dans le tore")
plt.yticks([0])
plt.grid()

plt.show()

## r(t):
plt.figure()
plt.plot(t, sol[:, 0], label='r(t)')
plt.legend(loc='best')
plt.xlabel('t',fontsize=30)
plt.ylabel('r(t)',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
#plt.xlim(0,10**-6)
plt.tight_layout()
plt.show()

## r(t) (avec horizontales):
x1p,x2p=zone_tore(f,R,p_dicho)
print(x1p,x2p)

plt.figure()
Y1=[x1p for i in range(len(sol[:, 0]))]
Y2=[x2p for i in range(len(sol[:, 0]))]
plt.plot(t, sol[:, 0], label='r(t)')
plt.plot(t,Y1,"red")
plt.plot(t,Y2,"red")
plt.legend(loc='best')
plt.xlabel('t',fontsize=30)
plt.ylabel('r(t)',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.tight_layout()
plt.show()

## z(t):
plt.figure()
plt.plot(t, sol[:, 3], label='z(t)')
plt.legend(loc='best')
plt.xlabel('t',fontsize=30)
plt.ylabel('z(t)',fontsize=30)
plt.xticks([0*10**-4,0.5*10**-4,10**-4],fontsize=20)
plt.yticks(fontsize=20)
plt.tight_layout()
plt.show()

## z(t) (réduit):
plt.figure()
plt.plot(t, sol[:, 3], label='z(t)')
plt.legend(loc='best')
plt.xlabel('t',fontsize=30)
plt.ylabel('z(t)',fontsize=30)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.xlim(0,10**-7)
plt.ylim(0,2.5*10**-5)
plt.tight_layout()
plt.show()

## Négatif <=> Sortie du tore:

plt.figure()
tore=[rho**2 -(sol[i,0]-R)**2-sol[i,3]**2 for i in range(len(sol))]

plt.plot(t,tore)
plt.xticks([0,5*10**-5,10**-4],fontsize=20)
plt.yticks(fontsize=20)
plt.grid()
plt.tight_layout()
plt.show()

```

```
##Négatif <=> Sortie du tore (réduit):
plt.figure()
plt.plot(t,dans_tore,label="positif<=>particule dans le tore")
plt.plot(t,[0 for i in range(len(t))],color="red")
plt.plot([T_collision for i in range(10)],[i-1 for i in range(10)],color="red")
plt.yticks([0],fontsize=20)
plt.xticks([0,5.4*10**-11,6*10**-11],fontsize=20)
#plt.xticks(fontsize=20)
plt.xlim(0,6*10**-11)
plt.ylim(-0.1,0.5)
plt.grid()
plt.tight_layout()
plt.show()

## Tore:

def pol_cart(list):
    r,theta,z = list[0],list[2],list[3]
    x=r*cos(theta)
    y=r*sin(theta)
    return(x,y,z)

def l_pol_cart(sol):
    X,Y,Z=[],[],[]
    for i in range(len(sol)):
        x,y,z=pol_cart(sol[i])
        X.append(x)
        Y.append(y)
        Z.append(z)
    return(X,Y,Z)

X,Y,Z=l_pol_cart(sol)
tore = plt.figure("Tore",clear=True)
ax = tore.add_subplot(1, 1, 1, projection='3d')

(theta, phi) = np.meshgrid(np.linspace(0, 2 * np.pi, 41),
                           np.linspace(0, 2 * np.pi, 41))

x = (1.9 + np.cos(phi)) * np.cos(theta)
y = (1.9 + np.cos(phi)) * np.sin(theta)
z = 0.7*np.sin(phi)

dplot = ax.plot_surface(x,y,z,color='whitesmoke' ,alpha=0.5)
ax.set(xlabel='x',
       ylabel='y',
       zlabel='z',
       xlim = [-3.2, 3.2],
       ylim = [-3.2, 3.2],
       zlim = [-1, 1],
       xticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
       yticks = [-3.1,-2.4,-1.7,0,1.7,2.4,3.1],
       zticks = [-0.7, 0, 0.7],
       title='Tore')
ax.scatter(X, Y, Z, marker='p')
tore.tight_layout()
#ax.view_init(10, 45)
#ax.view_init(90,90)
ax.view_init(0,90)
plt.show()
##
for angle in range(0, 360):
    ax.view_init(angle, angle)
    plt.draw()
    plt.pause(.001)
```