チュートリアル

牧野真人

2020 年 9 月 26 日

1　はじめに

本文書は (磁石) 振り子のプログラムである PEM のチュートリアルである。ここでは、２つの例をあげる。まずは、簡単な例として楕円体の自由回転をシミュレーションする。そのあと、磁石振り子のシミュレーションを例にする。このシミュレーションは磁石を持った質点の振り子として文献 [1] にある。

2　楕円体の自由回転

2.1　入力 UDF

中身のないファイルをまず作成する。ファイル名は、任意でもよいが、

```
ellipsoid.udf
```

とする。拡張子は udf とする。ファイルをテキストエディタで開き、

```
\include {"pem1_3_def.udf"}
```

と記述して、保存する。このファイルを GOURMET より開いて編集する。
データ名

```
unitParameter
```

以下には、PEM の単位としている長さ、質量、時間、電流の値をいれる。今回は、特に入力しないで進める。

```
body[]
```

では、剛体の情報をいれる。この後、設定する

```
simulation.pendulum[]
```

simulation.body[]

では、この body[] の id を指定して、利用することになる。まず、要素を一つ増やす。メニューの Edit の Insert an Array Element か Add an Array Element、あるいは、ショートカットキーとして、ctrl+i、ctrl+e で要素を増やすことが出来る。
　要素が増えたら

body[0].id

に

0

を入れる。id は他のものと重複しなければ、任意の整数で良い。body[] は複数の shape[] から構成する。もちろん一つでも良い。今回は、一つだけにする。shape[] に要素を一つ増やす。

body[0].shape[0].center.x
body[0].shape[0].center.y
body[0].shape[0].center.z

は、[0,0,0] を入れる。粒子固定座標系での回転の中心が原点であり、この shape[0] の中心が回転の中心となる。この後、入力する楕円体の中心が回転の中心になる。また、ここで入力している座標は、粒子の固定座標系で $\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3$ を基底とする座標系を入力していることになる。

body[0].shape[0].mass

は

1

とする。質量を入力した。

body[0].shape[0].shape

は

ellipsoid

を選択する。

body[0].shape[0].ellipsoid.a
body[0].shape[0].ellipsoid.b
body[0].shape[0].ellipsoid.c

には、楕円体の径の長さを指定する。

body[0].shape[0].ellipsoid.da.x
body[0].shape[0].ellipsoid.da.y
body[0].shape[0].ellipsoid.da.z

および

body[0].shape[0].ellipsoid.db.x
body[0].shape[0].ellipsoid.db.y
body[0].shape[0].ellipsoid.db.z

には、径 a, b に相当する単位ベクトルを入れる。楕円体の３つの軸は直交するため、この２つの外積から、body[0].shape[0].ellipsoid.dc に相当するものは計算されるため入力を要求していない。

body[0].magnet[]

以下には、磁気ダイポール (磁石) の情報をいれるが、ここでは、考えない。

body[0].color[0].red
body[0].color[0].green
body[0].color[0].blue
body[0].color[0].trans

は、body[0].shape[0] に相当する色を赤緑青および透明度を０から１の値を入れる。また、body[0].shape[] の要素数と body[0].color[] の要素数が一致しない場合 body[0].color[0] の色で、すべての body[0].shape[] を body[0].color[0] の色で描画する。

body[0].analysis

以下は、body[0].shape[] の値を元に解析した結果が入る。解析は、データ名

body[0]

の上を右クリックして、action のメニューを出して

analyzer

をクリックして解析する。この場合は、body[0] のみを解析するが、

analyzer_all

をクリックすると、すべての body[] を解析する。

body[0]

の形状、磁気ダイポールをプロットするには、action の

show....

を選択すれば良い。
　この body[0] を配置して、シミュレーションを行う。

simulation.time.simulationSteps
simulation.time.reportSteps
simulation.time.dt

は、シミュレーションのステップ数、データ出力のステップ間隔、ステップの間隔における時間 dt を入れる。ここでは、100000,100,0.001 を入れた。

simulation.systemSize.min.x
simulation.systemSize.min.y
simulation.systemSize.min.z
simulation.systemSize.max.x
simulation.systemSize.max.y
simulation.systemSize.max.z

¥は、系の x, y, z の最小値、最大値を指定することにより系のサイズを指定する。この後の

simulation.periodicBoundaryCondition

を true として、境界を超えて、磁気ダイポールが相互作用する場合は、系のサイズが意味を持つが、それ以外は、描画の際に、立方体の枠を描画する以外に意味はない。

simulation.integrator

は、Euler または 4thOrderRungeKutta が選択できる。精度の良い 4thOrderRungeKutta を選ぶのを勧める。

simulation.periodicBoundaryCondition

は、false とする。今回は、一体しかないこと、加えて、磁気ダイポールを指定していないので true を選んでも結果は変わらない。
　次に、

simulation.pendulum[]

を入力する。こちらは、重力場、磁場、相互作用で回転する剛体を設定する。一方

`simulation.body[]`

には、固定した磁石や一定速度、一定角速度で移動、回転する物体を設定できる。磁気ダイポールを持たない物体も設定できるが、描画する以外には、意味を持たない。ここでは、要素を追加して、

`simulation.pendulum[0]`

とする。

`simulation.pendulum[0].body`

には、

`body[0].id`

で入力した `0` を入力する。

`simulation.pendulum[0].position`

には、

`body[0]`

の粒子固定座標系の原点を実験室系の座標を指定して配置する。ここでは、`0,0,0` としている。

`simulation.pendulum[0].orientation`

には、粒子固定座標系の $\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3$ が実験室系での基底ベクトル $\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$ で表した初期座標を入力することで粒子の向きを指定する。すなわち $\boldsymbol{u}_1 \cdot \boldsymbol{e}_x, \boldsymbol{u}_1 \cdot \boldsymbol{e}_y, \boldsymbol{u}_1 \cdot \boldsymbol{e}_z$ などを入れていることになる。ここでは、初期条件として粒子固定座標系と実験室系が並行として、`(1,0,0)(1,0,0),(0,0,1)` を入力した。

`simulation.pendulum[0].angularVelocity`

には、初期の角速度を入力する。適当な値を入れて自由回転させる。

`simulation.pendulum[0].constraint`

は回転方向を固定する場合に用いる。今回は、用いない。以下、

```
simulation.body[]
simulation.gravity
simulation.magnetiField
simulation.freeSpacePermeability
simulation.PseudoFrictionTorque
```

で、固定された物体、重力、外磁場、透磁率、摩擦トルクが入力できるが、特に、今回は考えない。以上、設定できたところで、ファイルを保存する。

2.2 シミュレーションの実行、結果

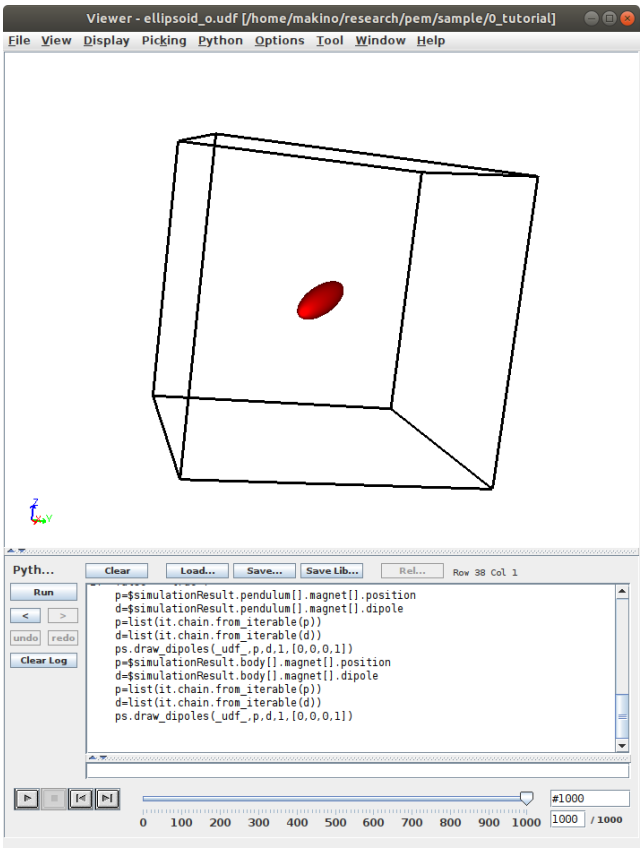

図 1: 楕円体の自由回転の描画

gourmetterm から

```
pem -I elllipsoid.udf -O ellipsoid_o.udf
```

のように作成した入力ファイルと出力ファイルを指定して、実行する。出力ファイルを設定しなければ、入力ファイルに結果が出力される。

計算した

`ellipsoid_o.udf`

を GOURMET から開く。データ名

`simulationResult`

を右クリックして、

`show...`

で、シミュレーション結果を描画できる。適当な視点で、動画を見ることが出来る。

3　磁石振り子

文献 [1] の１５２ページにある磁石振り子をシミュレーションする。この文献のシミュレーションの枠組みでは、図 2(a) のように、振り子の軸のまわりに対称である必要があるが、PEM では、図 2(b) のように対称である必要ない。

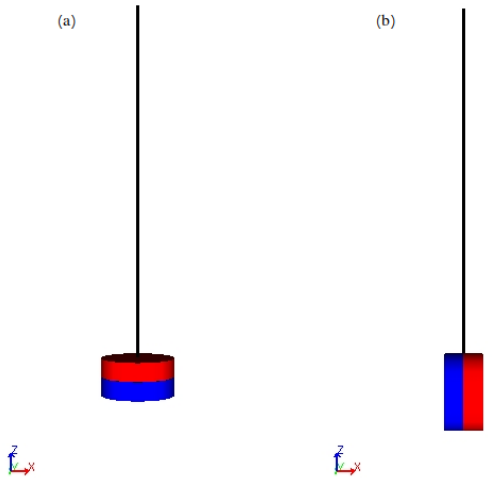

図 2: 振り子の例。(a) は軸のまわりで対称。(b) は非対称な場合。

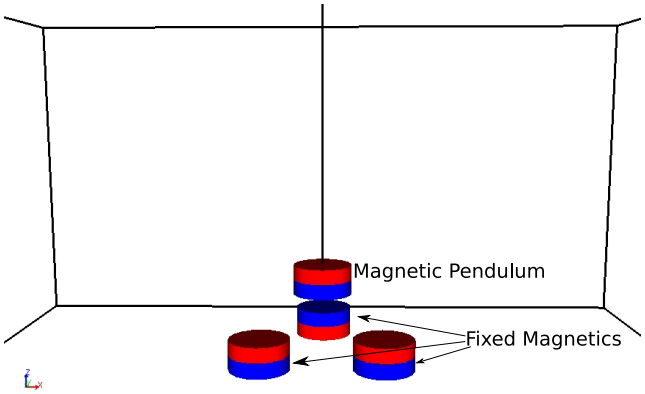

図 3: 磁石振り子。固定された磁石があり、相互作用することにより複雑な運動をする。

3.1 入力 UDF

図 3 は、目標とするシミュレーションである。

赤色を磁石の N 極、青色を S 極として描画している。磁石の振り子があり、下に固定された磁石と相互作用することにより、振り子は複雑な運動をする。

ここでは、

```
body[0]
body[1]
```

と 2 つ設定する。片方が、振り子で、片方が下に設置する磁石である。まず磁石の振り子を

```
body[0]
```

に設定する。

```
body[0].shape[0]
body[0].shape[1]
```

に円筒で色を変えて N 極と S 極を表現している。粒子固定座標系の原点が回転中心なので、円筒の長さを考えて、配置している。

```
body[0].shape[2 2]
```

は描画用に、直線を設定している。質量がゼロであるから、物理的には意味をなさない。

```
body[0].magnet[0]
```

には、振り子の磁石を設定している。

`body[1]`

では、下に固定する磁石の形状を設定している。`body[0]` のものと似てはいえるが、振り子の軸の部分がなく、原点に磁石の中心がある。

`simulation.pendulum[0]`

に、振り子を設定する。回転中心を実験室系の原点においている。初期速度を `simulation.pendulum[0].angularVelocity` に設定して運動するようにしている。

`simulation.body[0]`
`simulation.body[1]`
`simulation.body[2]`

で磁石を設定している。特に

`simulation.body[0].orientation`

は `simulation.body[0].orientation.u3.z` を `-1` として、磁石を反転している。また、右手系を保つために `simulation.body[0].orientation.u2.y` も `-1` にしている。

`simulation.gravity`

に重力を設定して、入力 udf が出来た。

3.2 シミュレーションの実行、結果

gourmetterm から

```
pem -I magnetic_pendulum.udf -O magnetic_pendulum_o.udf
```

のように実行。出力されるファイルを GOURMET で開いてアニメーションを見ると、複雑な運動をしているのが分かる。例えば、下の python を gourmetterm から実行すると図 4 のように軌跡が分かる。ただし matplotlib を pip でインストールしておく必要がある。

```
import numpy as np
from UDFManager import UDFManager
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

udf=UDFManager("pendulum_o.udf")
pos=[]
n=udf.totalRecord()
for i in range(n):
    udf.jump(i)
    p=np.array(udf.get("simulationResult.pendulum[0].magnet[0].position"))
    pos.append(p)
pos=np.array(pos)
#
fig = plt.figure(figsize=(9,5))
ax = fig.add_subplot(111, projection='3d')
ax.plot(pos[:,0],pos[:,1],pos[:,2],color="blue")
ax.scatter(pos[::10,0],pos[::10,1],pos[::10,2],color="black")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
ax.set_xlim(-4,4)
ax.set_ylim(-4,4)
ax.set_zlim(-12,2)
ax.set_title("Bob trajectory")
#
plt.tight_layout()
plt.savefig('./trajctory.png')
plt.show()
```

たとえば、磁石の向きを変えたものが、

```
magnetic_pendulum2.udf
```

である。振り子の軸で対称でない場合もシミュレーションできる。

実際の実験では、摩擦があるため、最終的に振り子は、どこかで止まってしまう。udf の

```
simulation.PseudoFrictionTorque
```

を有効にしているのが、

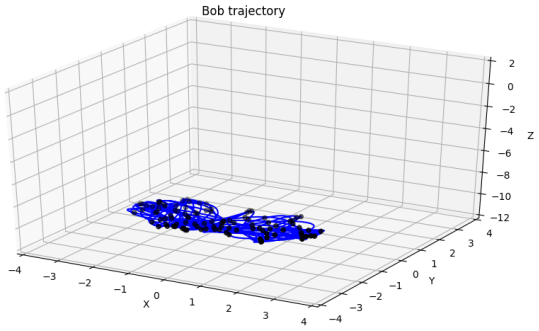

図 4: 磁石振り子の磁石の位置の軌跡

`magnetic_pendulum_friction.udf`

である。十分に時間が経った結果、図 5 に、N 極と S 極とが引き合う位置が平行位置として計算されており、最終的な位置は、適切と考えられる。しかし、摩擦が振り子の向きに関係なく等方的な摩擦が入っているなど、動力学には、妥当性はない。

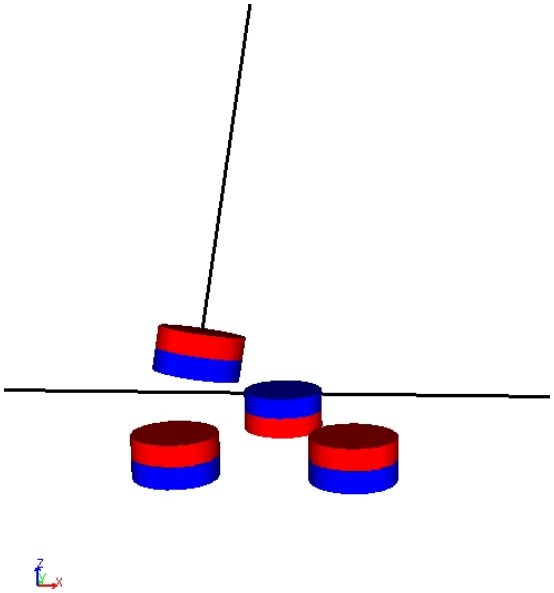

図 5: 平衡時の振り子の位置

参考文献

[1] 土井正男、滝本淳一編 物理仮想実験室　名古屋大学出版　(2004)