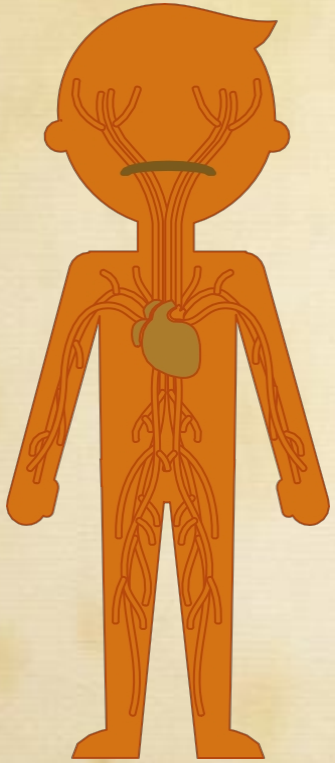
Wireless transmission of electrical power: biomedical implants

2021-2022 THEME:
HEALTH AND PREVENTION

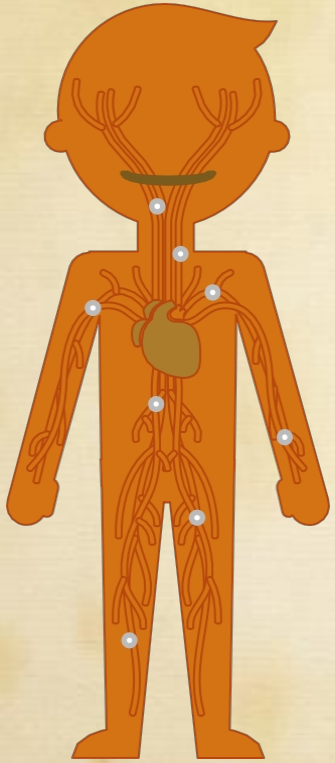
ADHAR Emna

MOTIVATION



The situation is real: you want to recharge your patient's cardiac implant and you want to spare him the surgery.

OBJECTIVE



Charging the battery by

INDUCTION

CONTENTS

01

NON-RESONANT COUPLING

02

RESONANT COUPLING

03

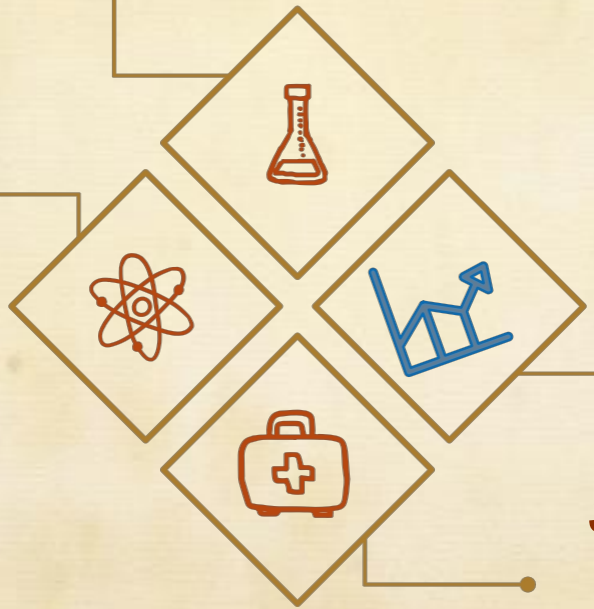
DEVELOPING MY OWN MODELS TO IMPROVE PERFORMANCE

- Digital model 
- Experimental model 

WORKING METHOD

experiment

Theoretical approach

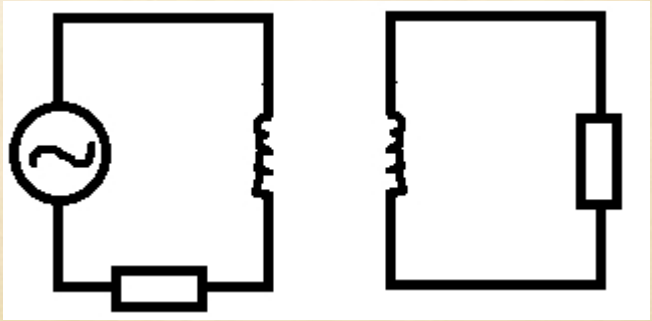


Results

Seeking to improve results

01

NON-RESONANT COUPLING

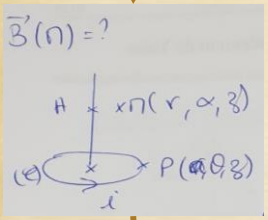


THEORETICAL APPROACH

- BIOT-SAVART Law
- Faraday's Law
- Calculating yield: is the Yates formula valid?

Biot-savart law

$$\overrightarrow{B(M)} = \int_{P\in(C)} \frac{\mu_0}{4\pi} i\,\overrightarrow{dl} \wedge \frac{\overrightarrow{PM}}{PM^3}$$



$$\overrightarrow{B(M)} = \frac{\mu_0 * i}{4\pi * ((r-a)^2 + z^2)^{3/2}} * (az\,\overrightarrow{er} + (a-r) * a\,\overrightarrow{ez})$$

Factors to consider during the experiment

01 Coil shape issuer

02 Current flowing in the circuit primary

03 ...

04 ...

05 ...

06 ...

Faraday's Law

$$e = -\frac{d\emptyset}{dt}$$

Factors to consider during the experiment

01 Coil shape issuer	**02** Current flowing in the circuit primary	**03** Position of the take-up reel
04 Coil shape receiver	**05** ...	**06** ...

Yield calculation

$$\eta = \frac{1}{R} * \frac{u_{Rm}^2}{u_{Gm}^2} * \sqrt{1 + \left(\frac{L_1 + M}{R}\omega\right)^2}$$

How do you calculate M?

$$M = \frac{1}{2\pi f}\sqrt{\frac{{u_{Gm}}^2}{2 * {I_{eff}}^2}} * L_1$$

Formula by Yates (to be verified)

$$\eta = k\frac{\mu_0{}^2 N_1{}^2 N_2{}^2 a^4 b^4 \omega^2}{R_1 R_2 (d^2 + a^2)^3}$$

- N_1 number of turns of the coil issuer
- N_2 Number of coil turns receiver
- a Radius of the transmitting coil
- b Radius of the receiving coil
- d Distance between the two coils

Factors to consider during the experiment

01 Coil shape issuer

AND SIZE!

02 Current flowing in the circuit primary

03 Position of the take-up reel

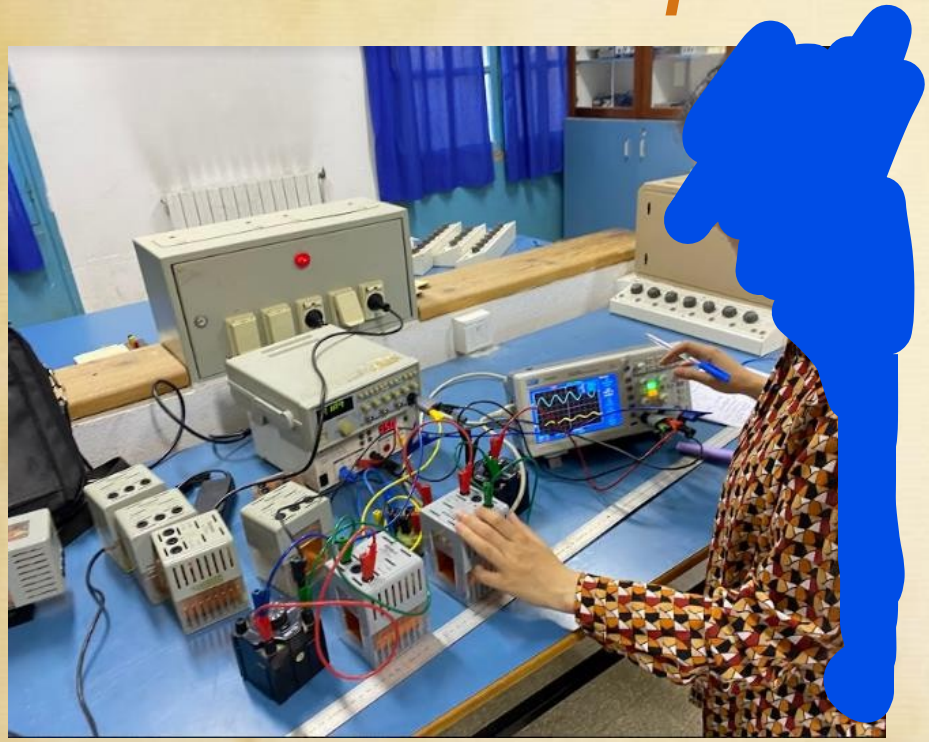
04 Coil shape receiver

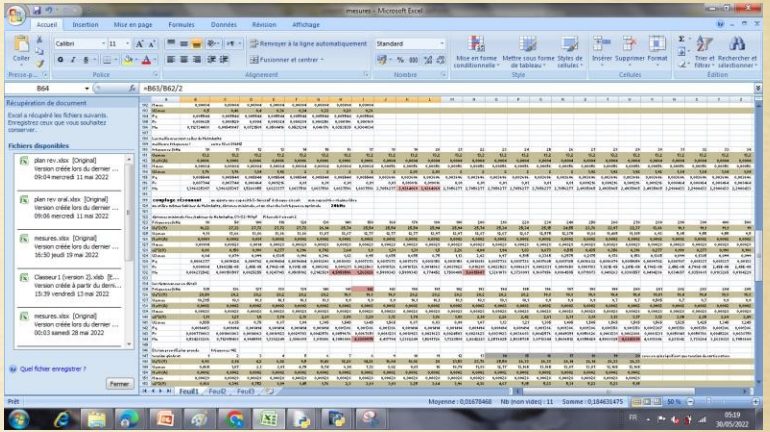
AND SIZE!

05 DON'T FORGET TO CONNECT AN AMMETER TO THE TRANSMITTER CIRCUIT!

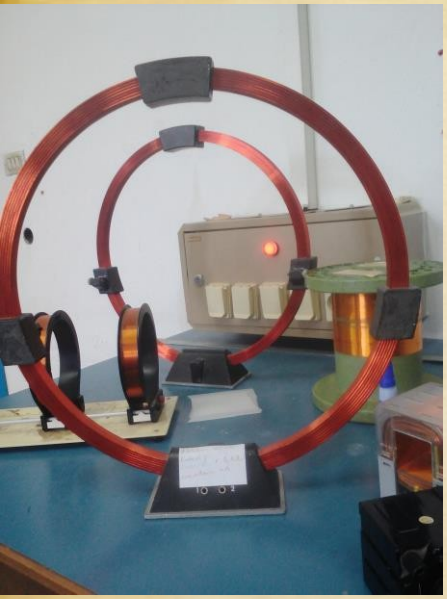
06 For better mutual inductance, different or similar coils?

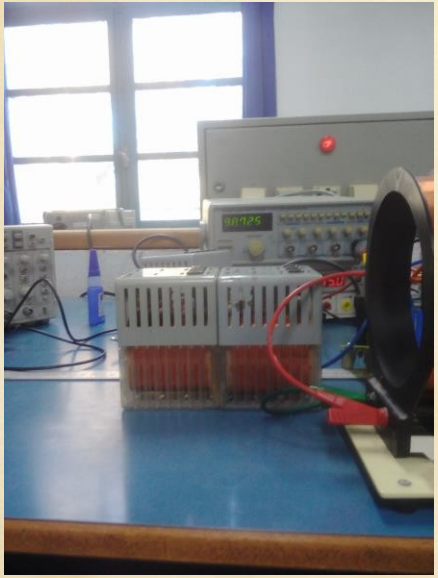
Carrying out the first series of experiments

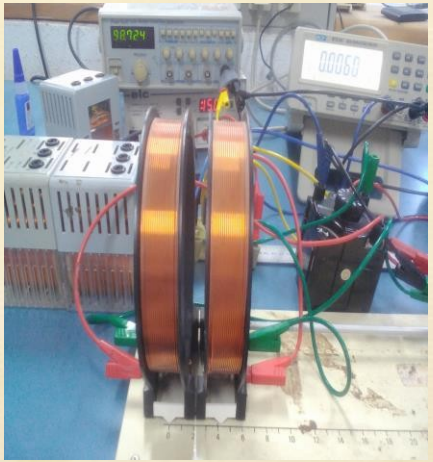


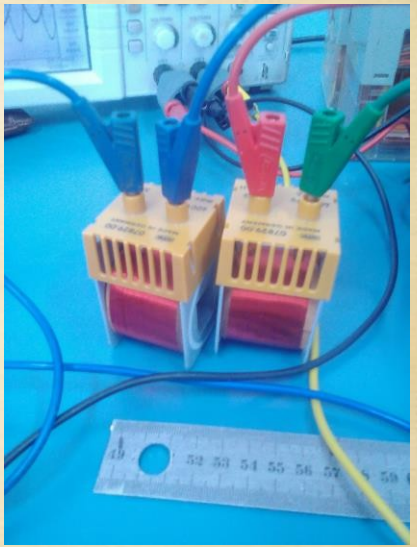


01 Coil shape issuer

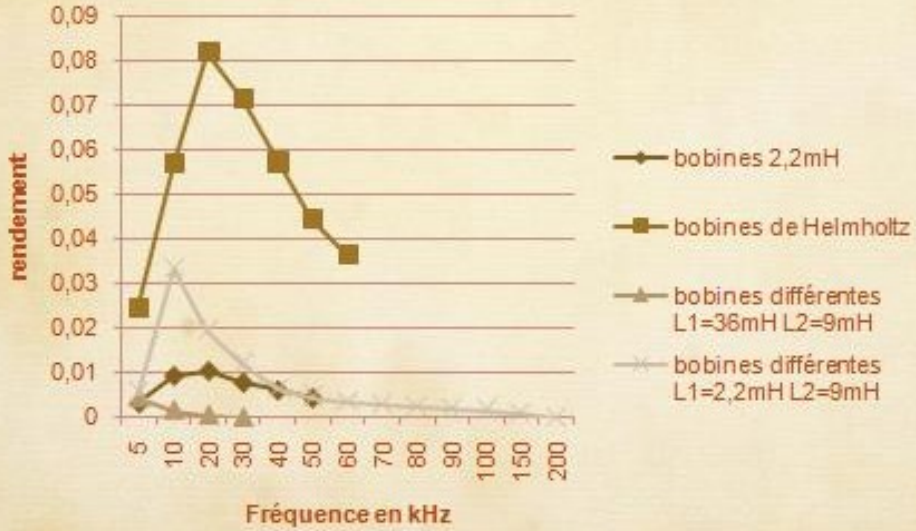








Plusieurs mesures pour différentes bobines à différentes fréquences



Formule de Yates n'est pas valide en Hautes fréquences: l'effet de peau

Factors to consider during the experiment

01 *Coil shape issuer*

Improved efficiency for Helmholtz coils

02 *Current flowing in the circuit primary*

Pushpull for amplification + choose a suitable frequency

03 *Position of the take-up reel*

04 *Coil shape receiver*

Having tried various possible combinations, the best are those of

05 *DON'T FORGET TO CONNECT AN AMMETER TO THE TRANSMITTER CIRCUIT!*

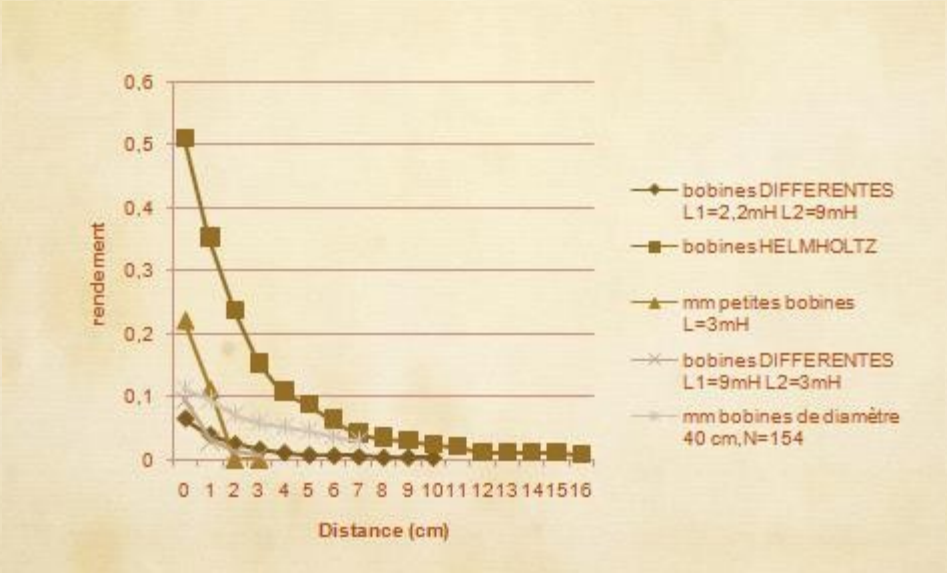
Helmholtz

06 *For better mutual inductance, different or similar coils?*

DONE! So we can calculate the mutual inductance and efficiency

Inductances are not necessarily similar.

Vary the distance between the coils=> test different types of coils positions of the take-up reel
(we work at the appropriate frequency for each assembly)



Factors to consider during the experiment

01 *Coil shape issuer*

Improved efficiency for Helmholtz coils

02 *Current flowing in the circuit primary*

Pushpull for amplification + choose a suitable frequency

03 *Position of the take-up reel*

It has a major influence on efficiency, and the best coils to withstand it are by far the Helmholtz coils.

04 *Coil shape receiver*

Having tried various possible combinations, the best are those of

05 *DON'T FORGET TO CONNECT AN AMMETER TO THE TRANSMITTER CIRCUIT!*

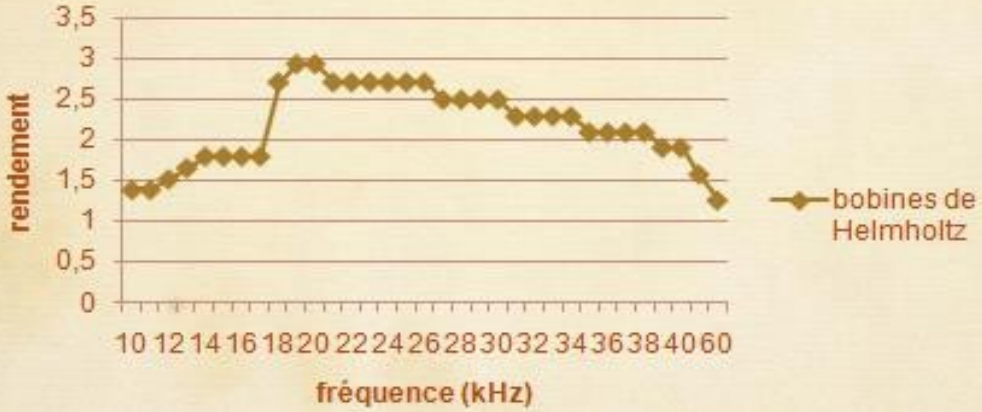
Helmholtz

06 *For better mutual inductance, different or similar coils?*

DONE! So we can calculate the mutual inductance and efficiency

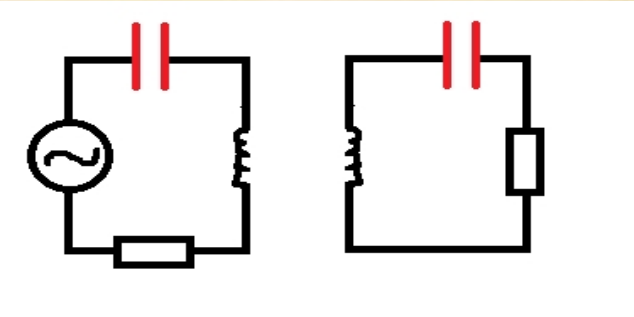
Inductances are not necessarily similar.

Helmholtz coils



02

RESONANT COUPLING



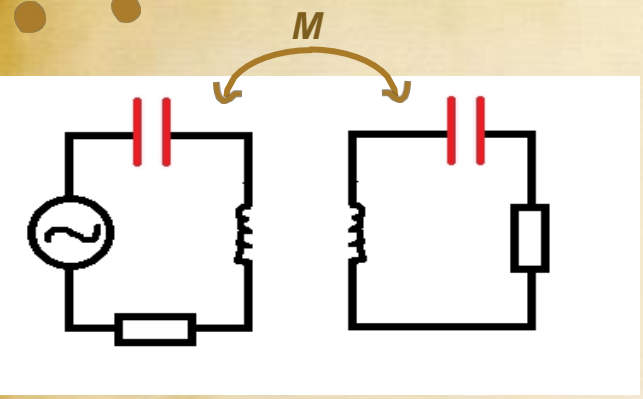
THEORETICAL APPROACH

- Why the two circuits be identical?
- Forced oscillations in harmonic regime, notion of resonance and antiresonance,
- Yield calculation

Should we choose identical circuits?

Demonstration annex 1

Resonance/anti-resonance



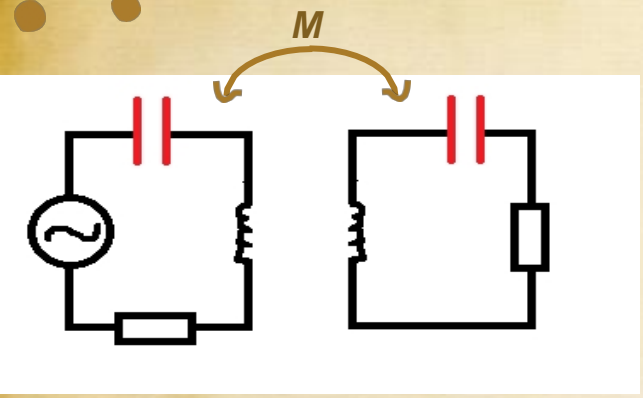
we apply the law of meshes, neglecting the resistancesC

$$\underline{i_1} = j\omega * \frac{k}{M} * \frac{\omega^2 - \omega_0^2}{\omega^4 k^2 - (\omega^2 - \omega_0^2)^2} * \underline{E}$$

$$\underline{i_2} = j\omega \frac{k^2 * \omega^2}{M} * \frac{1}{(\omega^2 - \omega_0^2)^2 - \omega^4 * k^2} \underline{E}$$

- Two pulses cancel out the denominator: the intensities then tend towards infinity and we observe the phenomenon of resonance.
- For $\omega = \omega_0$,i_1 cancels out, this is the anti-resonance pulsation

Resonance/anti-resonance



Note the two resonance pulses ω_d and ω_s

As they are the zeros of the polynomial:

$$P(X) = k^2X^4 - (X^2 - \omega_0^2)^2$$

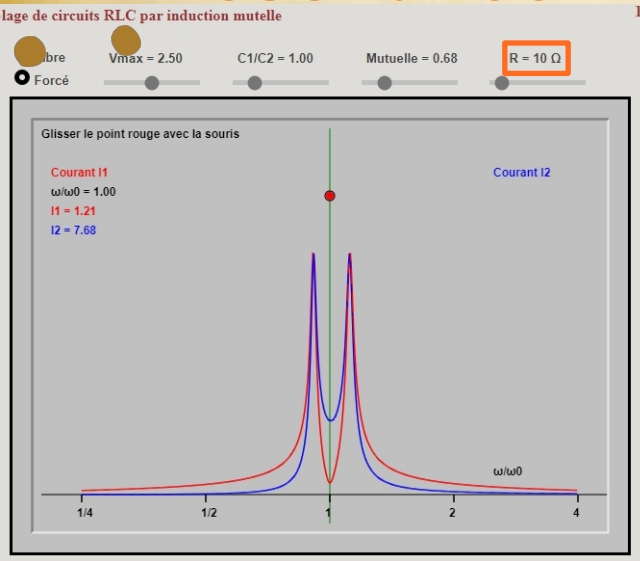
So we have:

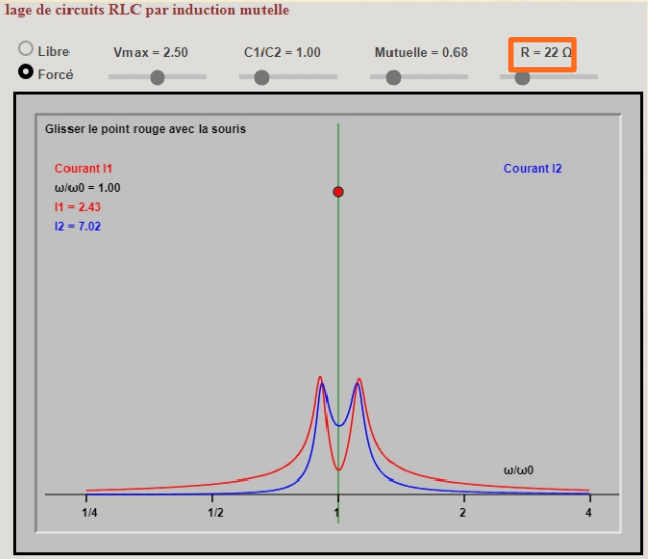
$$\omega_s\omega_d = \frac{{\omega_0}^2}{\sqrt{1-k^2}}$$

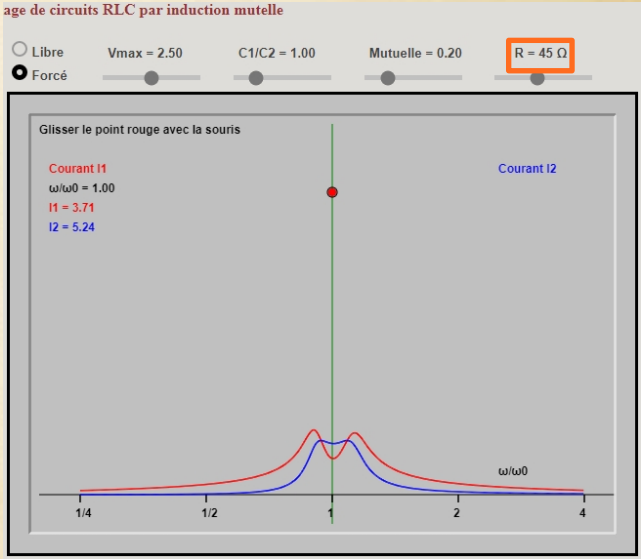
This results in a way of calculating k !

Resonance/ anti-resonance

Resistance
=> Attenuation of the acuity of the resonance







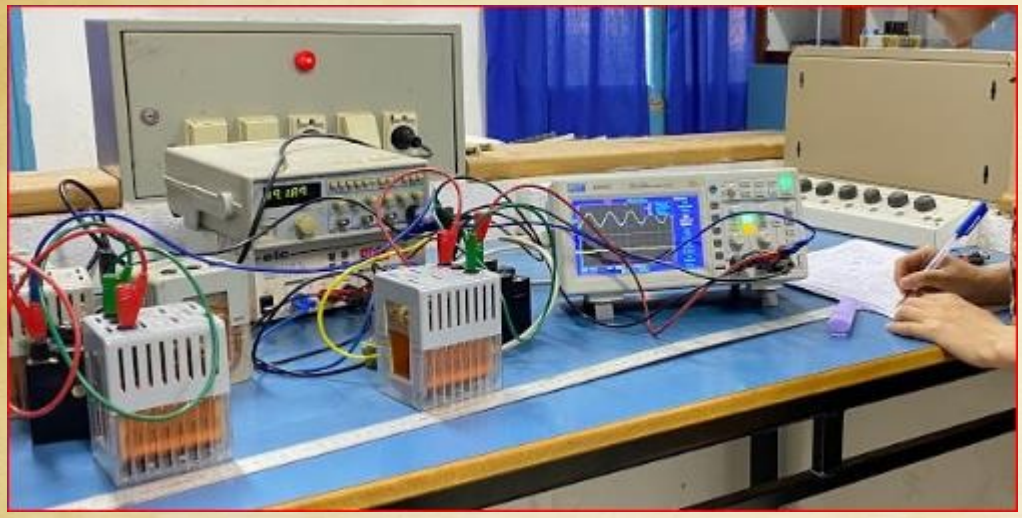
Simulation carried out by the site: Coupling of RLC circuits by mutual induction (univ-lemans.fr)

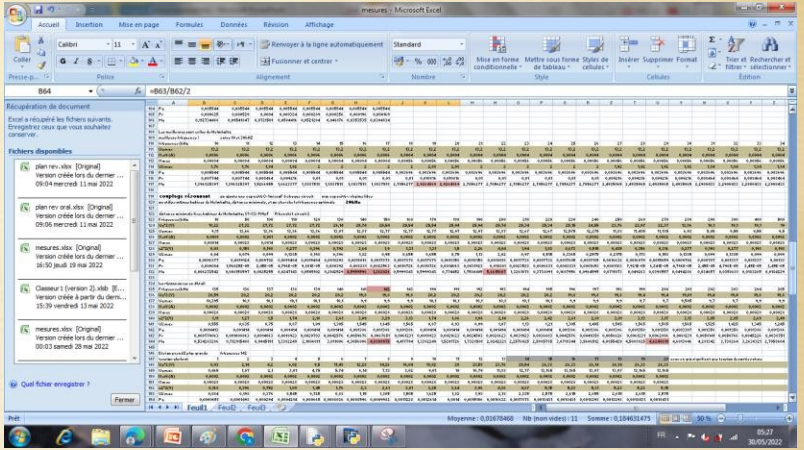
Calculating efficiency

The resistance is taken into account

$$\eta = \frac{1}{R^2} * \frac{{u_{Rmax}}^2}{E_{max}^2} * \frac{\left(R^2 - \left(\omega L - \frac{1}{C\omega}\right)^2 + (\omega M)^2\right) + 4R^2(\omega L - \frac{1}{C\omega})^2}{R^2 + (\omega M)^2 + (\omega L - \frac{1}{C\omega})^2}$$

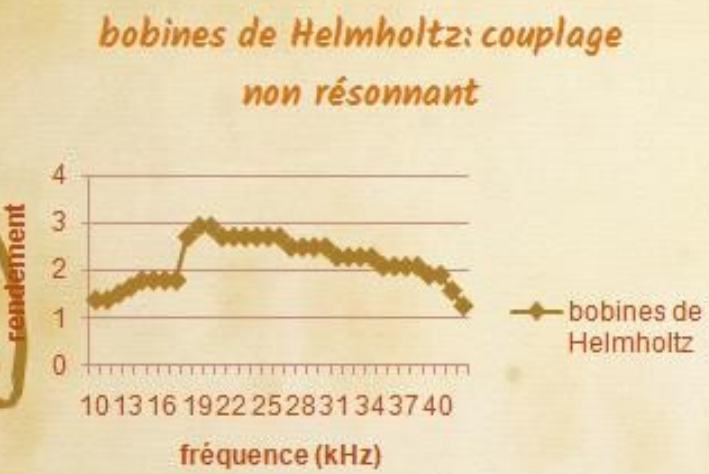
Second series of experiments



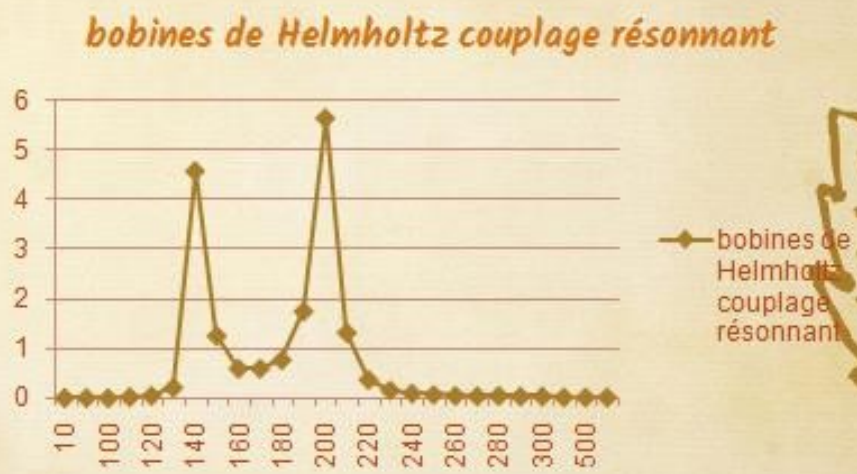


En ajoutant deux condensateurs...

Avant ...



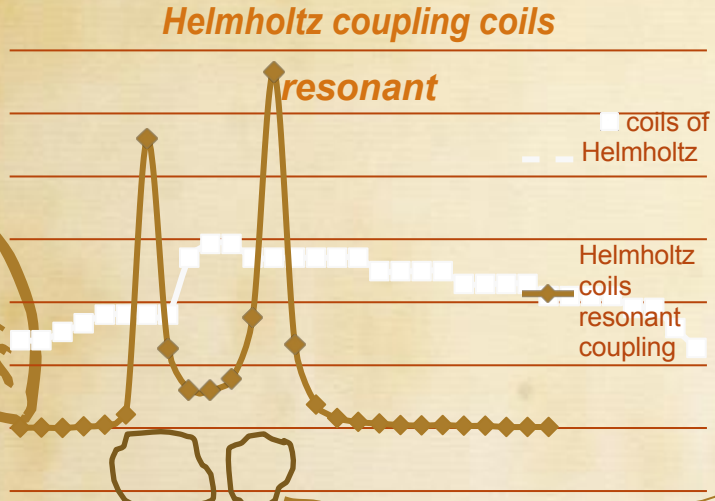
Après ...



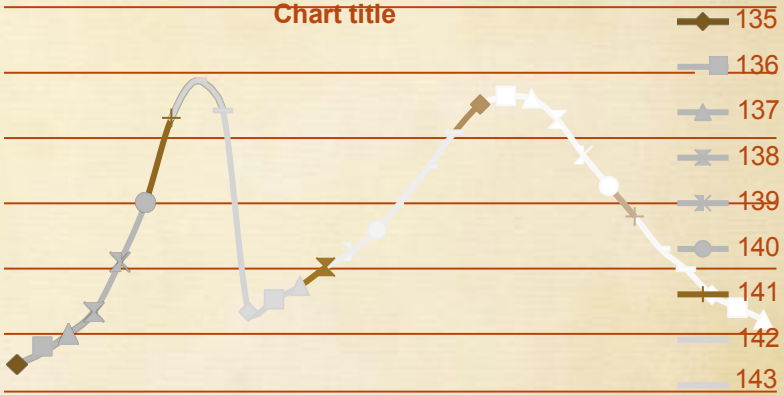
Quelle fréquence précisément ?

What frequencies exactly?

Previous measures...



Plus fin...



142kHz and 200kHz
=> √(wd*ws)≠w0 => coupling is not weak!

we fix ourselves in the optimum conditions we have found, and vary the distance between the two coils

What if we brought the metal plates of the coupled circuits closer together?

without plates

2V

WITH A SINGLE DIMENSION PLATE RECEIVER

1,52V

WITH ONE SIDE PLATE SENDER

1,92V

With two plates

2V

What does it mean? The birth of eddy currents

And now that we have an idea of the optimum conditions for resonant coupling, let's design our own model for cardiac implantation recharge!

03

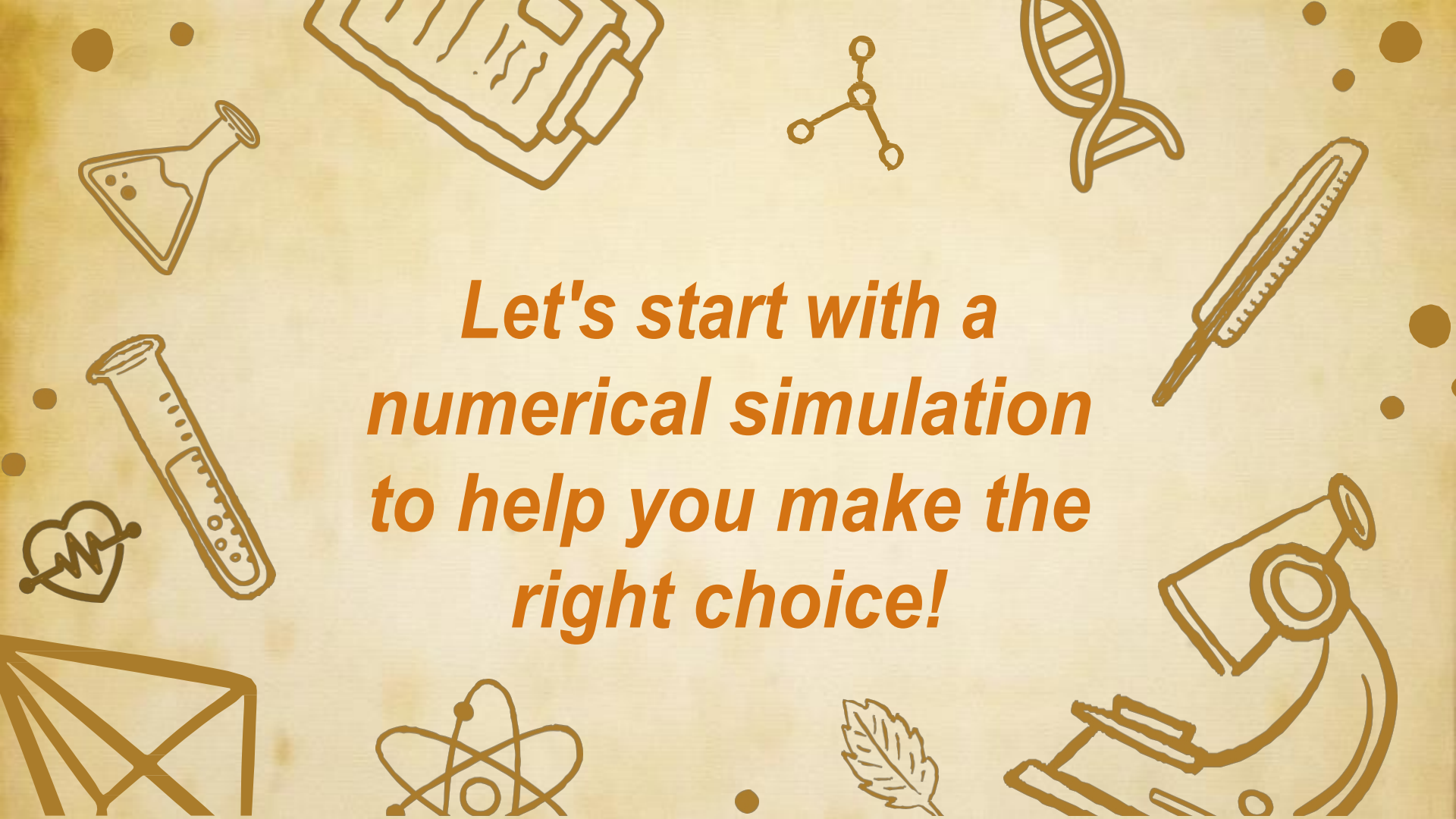
DEVELOP MY OWN MODELS TO IMPROVE PERFORMANCE

Model design

Results of previous experiments

The best coils for transferring electrical power were helmholtz coils, operating in resonant coupling, in the absence of any other metal.

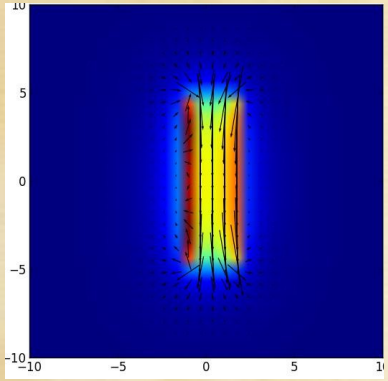
Factors to consider


Let's start with a
numerical simulation
to help you make the
right choice!

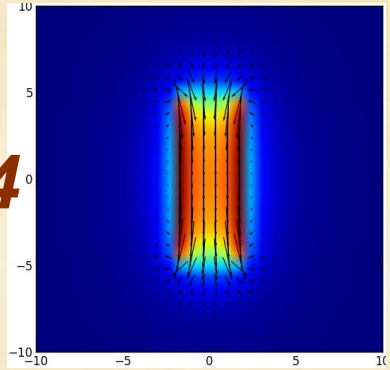
The BIOT-SAVART law has enabled me to write a Python code that gave born in .

Polygons with N ribs for which I have chosen radius=2cm and number of turns=100

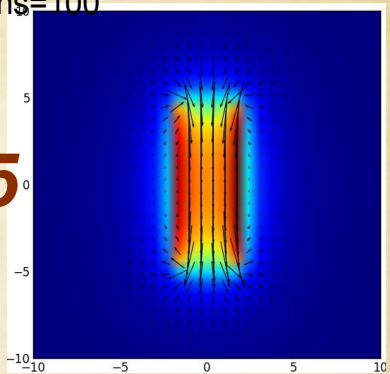
N=3



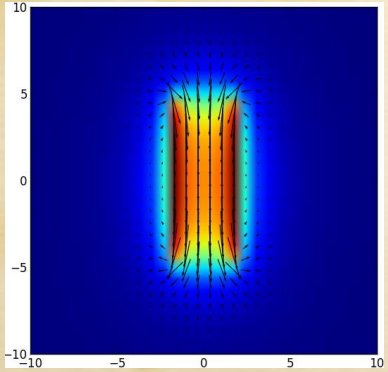
N=4



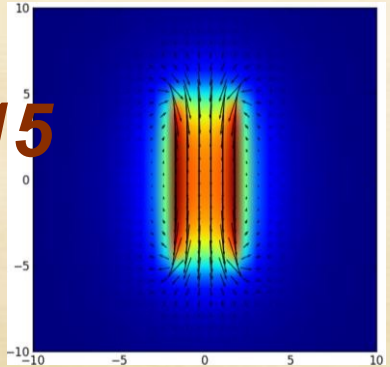
N=5



N=6

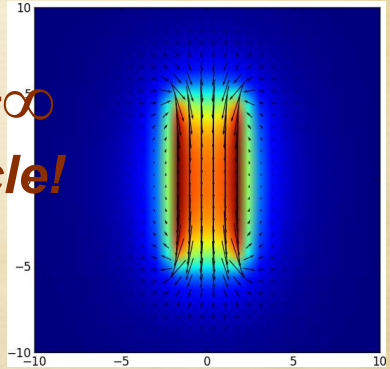


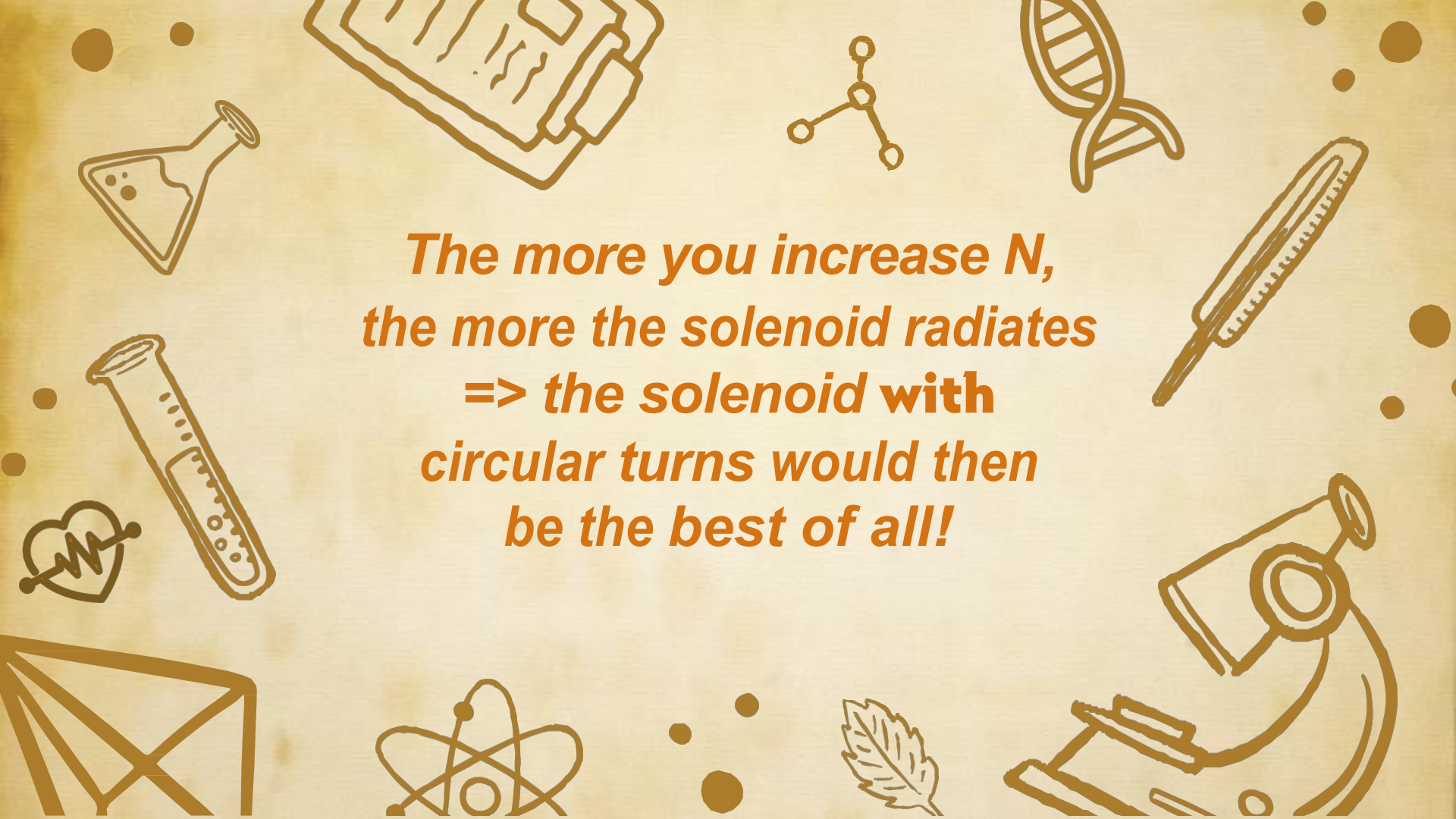
N=15



N=+∞

A circle!

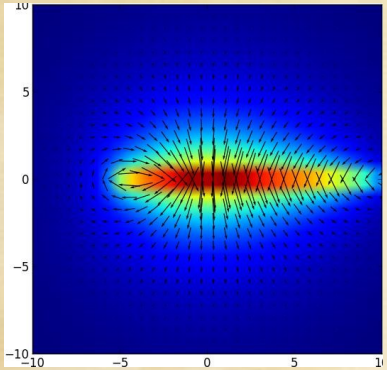



The more you increase N,
the more the solenoid radiates
=> the solenoid with
circular turns would then
be the best of all!

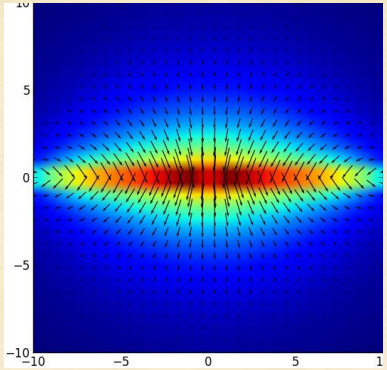
AND IN 2D?

Polygons with N sides for which I have chosen the number of turns = 100

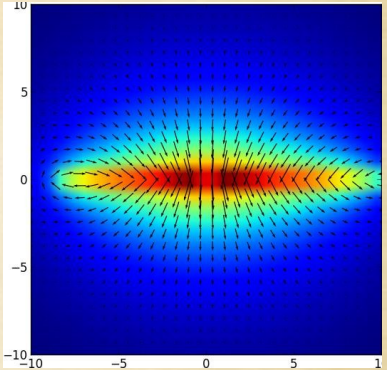
N=3



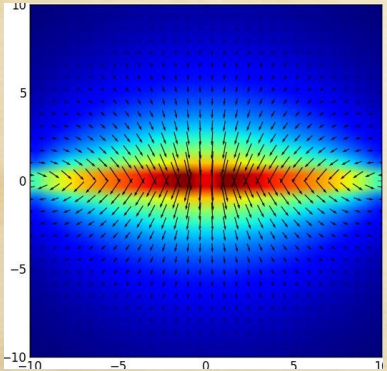
N=4



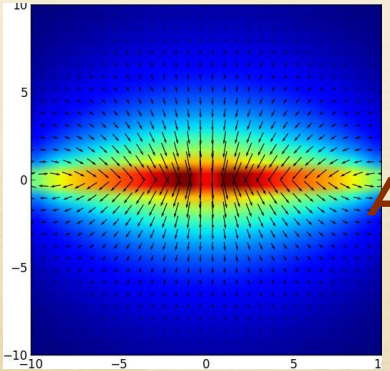
N=5



N=6

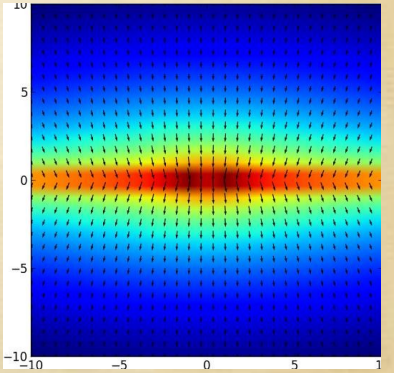


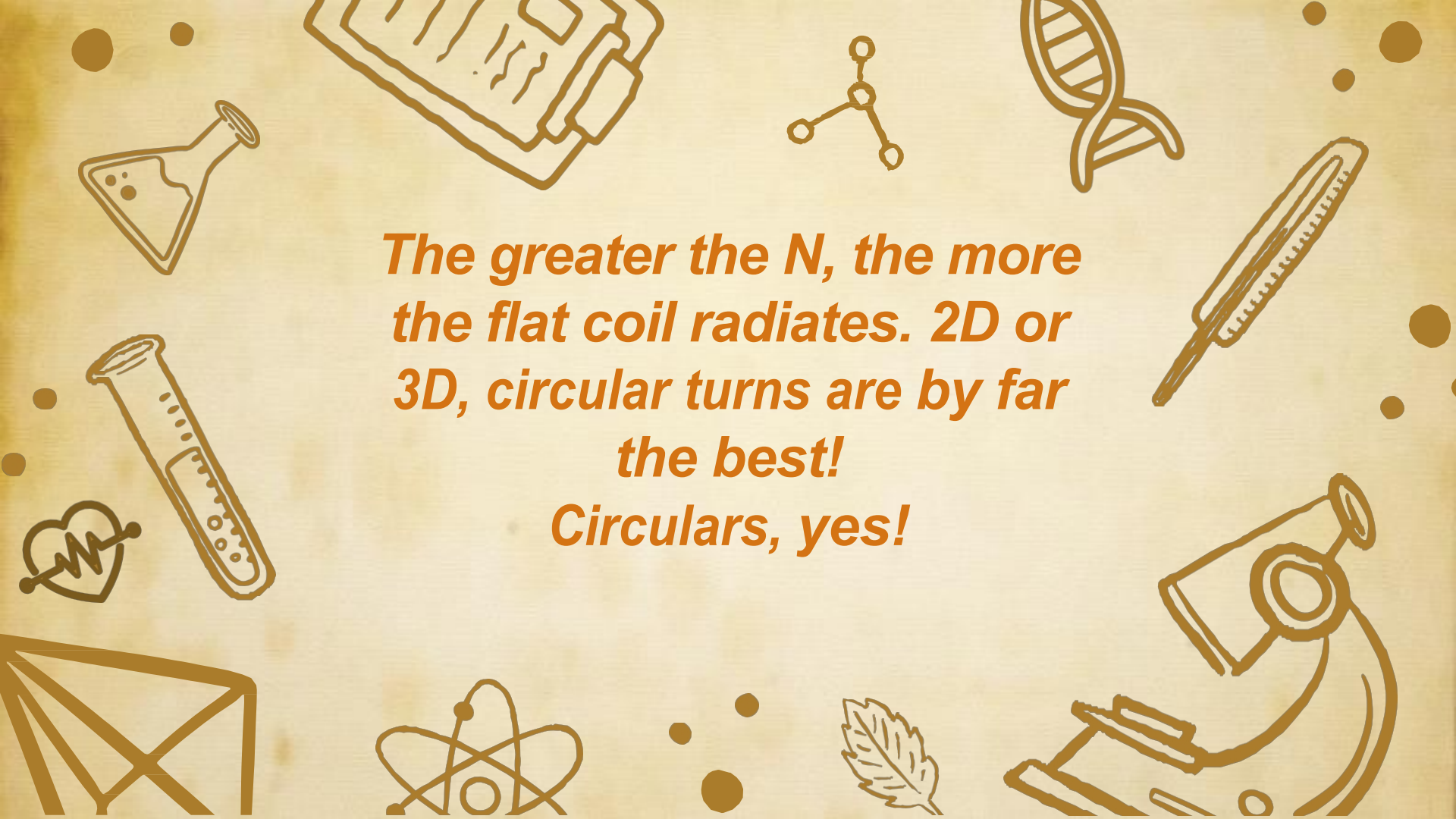
N=15



N=+∞

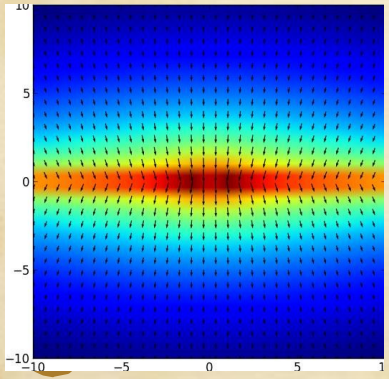
A circle!



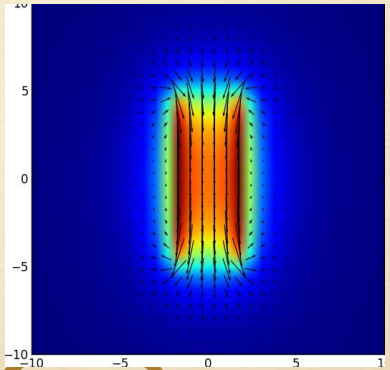

The greater the N, the more the flat coil radiates. 2D or 3D, circular turns are by far the best!
Circulars, yes!

2D or 3D? Flat or solenoid?

2D

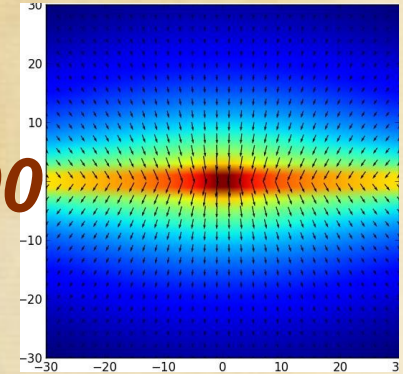


3D

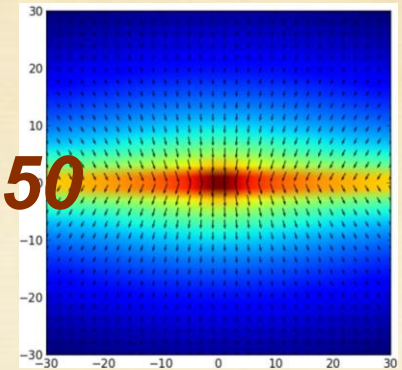


We choose flat coils with circular turns, of course. But how many turns?

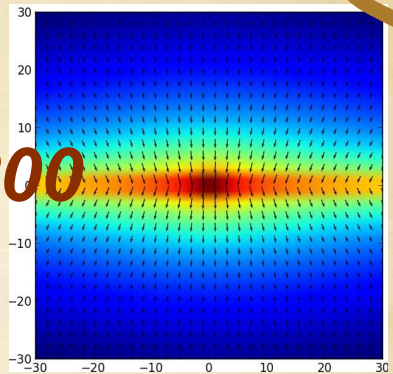
N=100



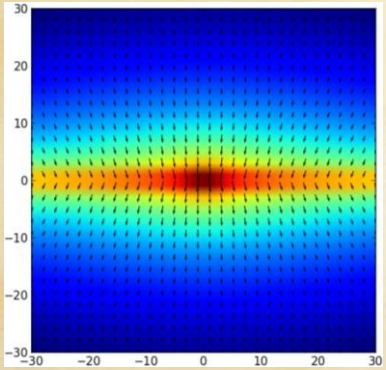
N=150



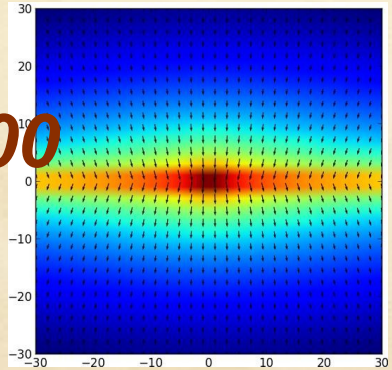
N=200



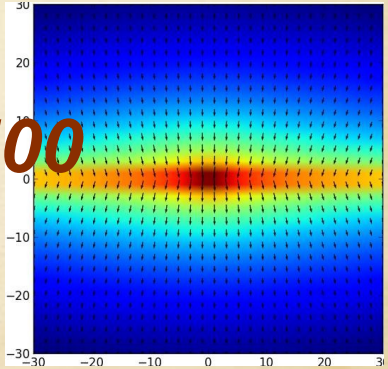
N=30



N=400



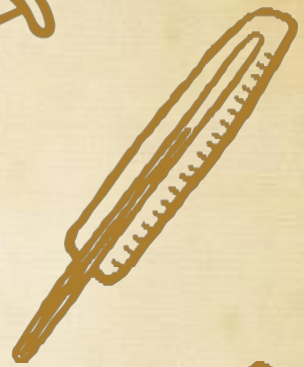
N=500

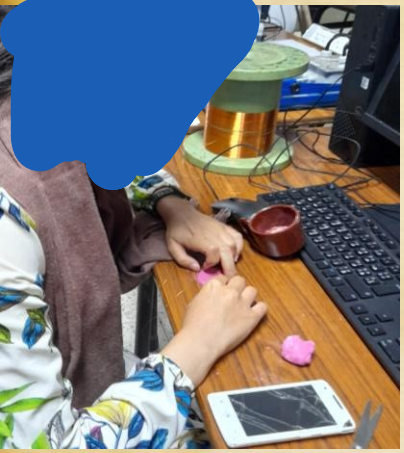


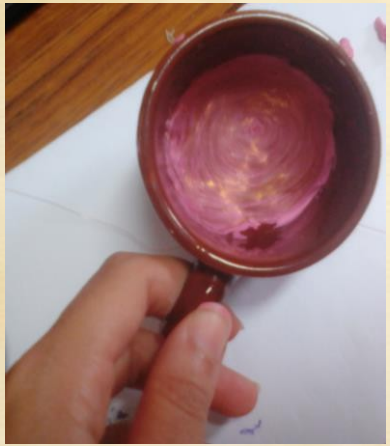
We don't have to increase the number of turns on our coil

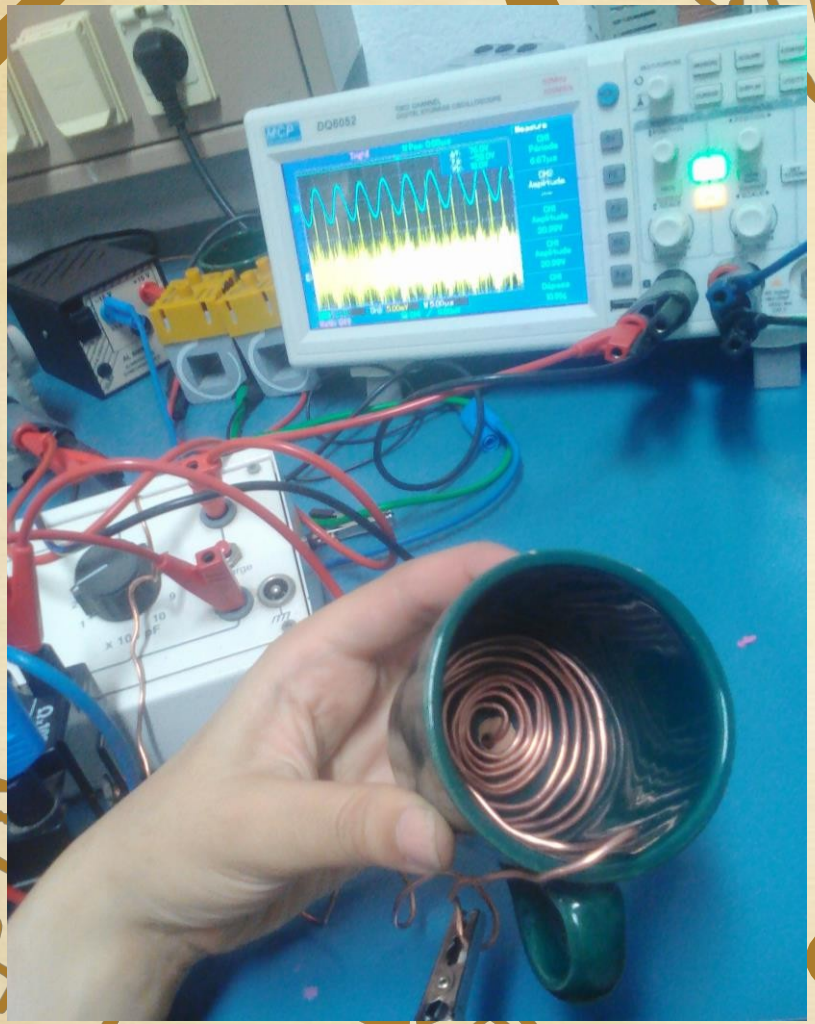
+ A ceramic semi-resonant cavity will eliminate radiation losses.

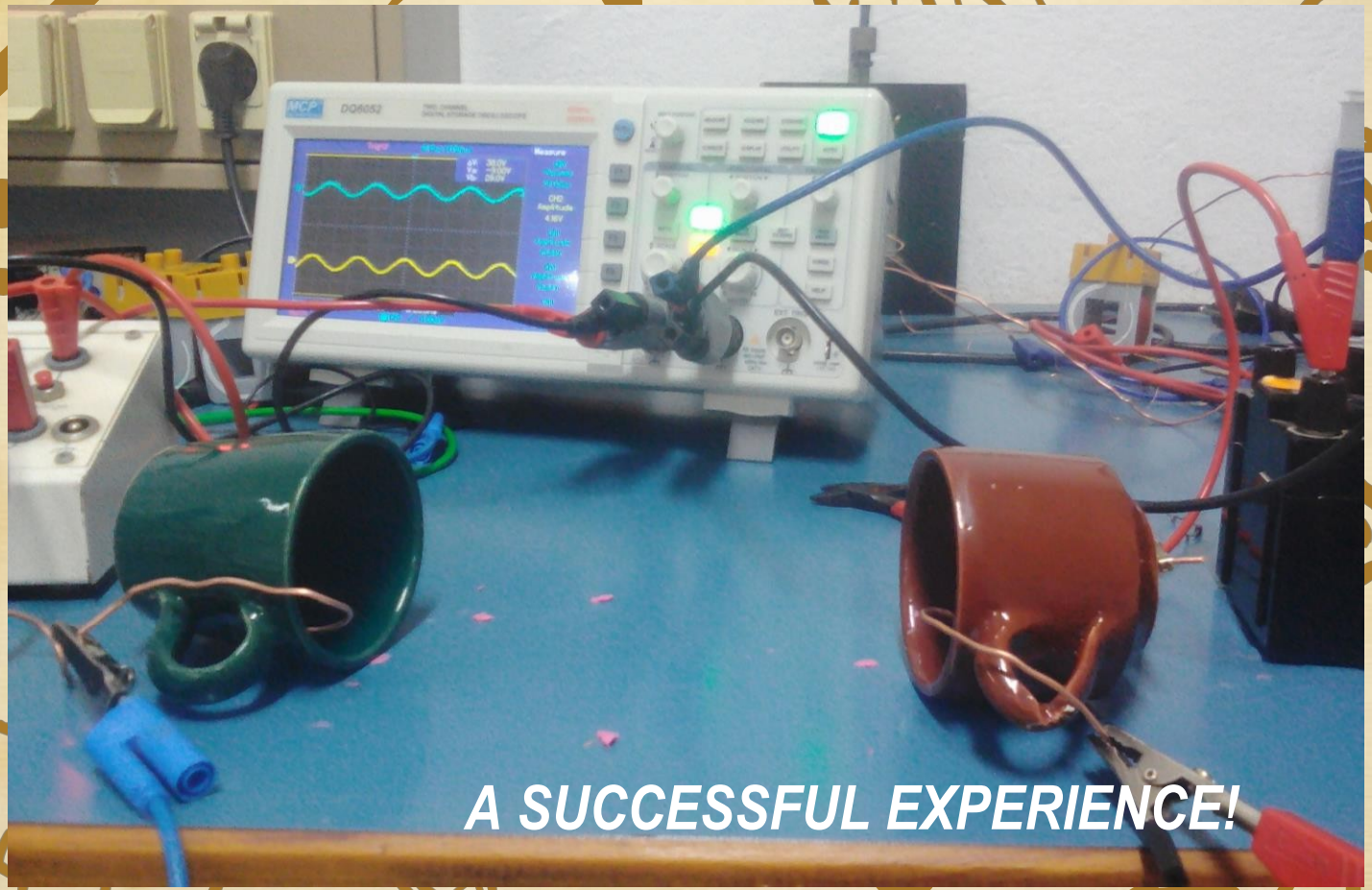
Now, to our tools!

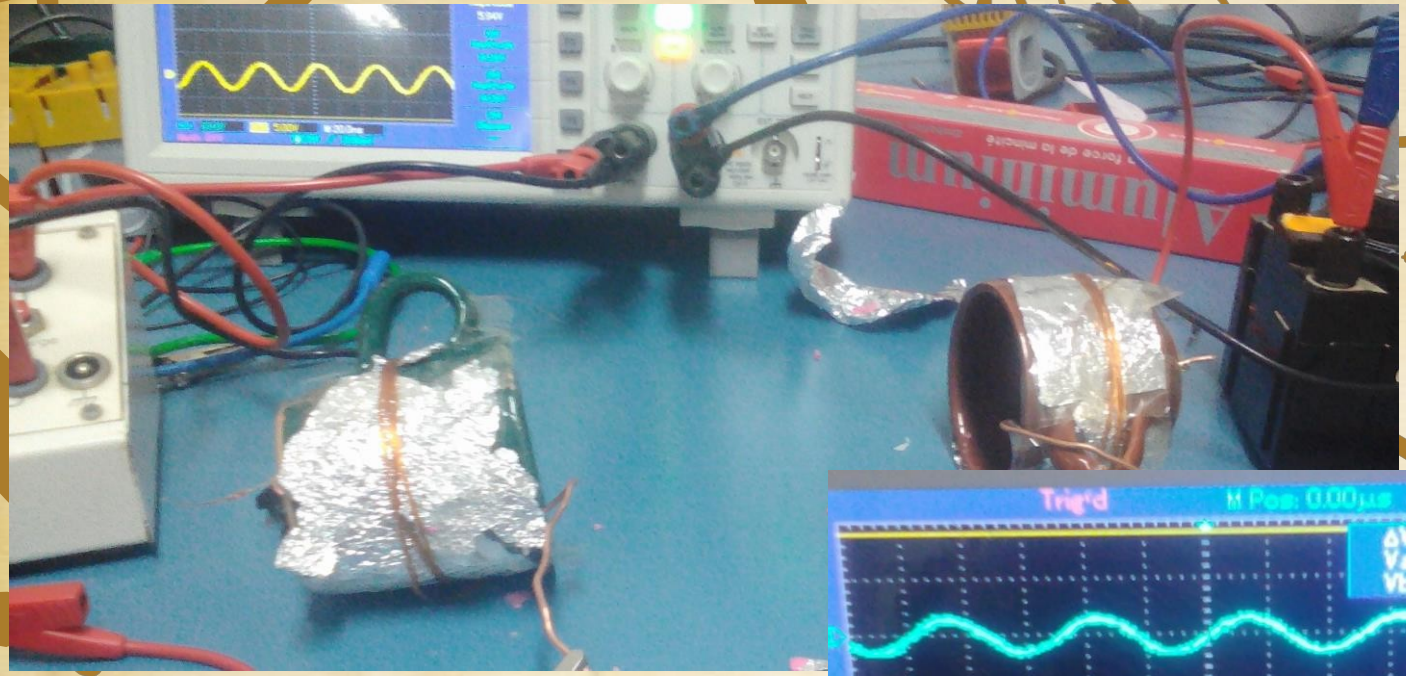


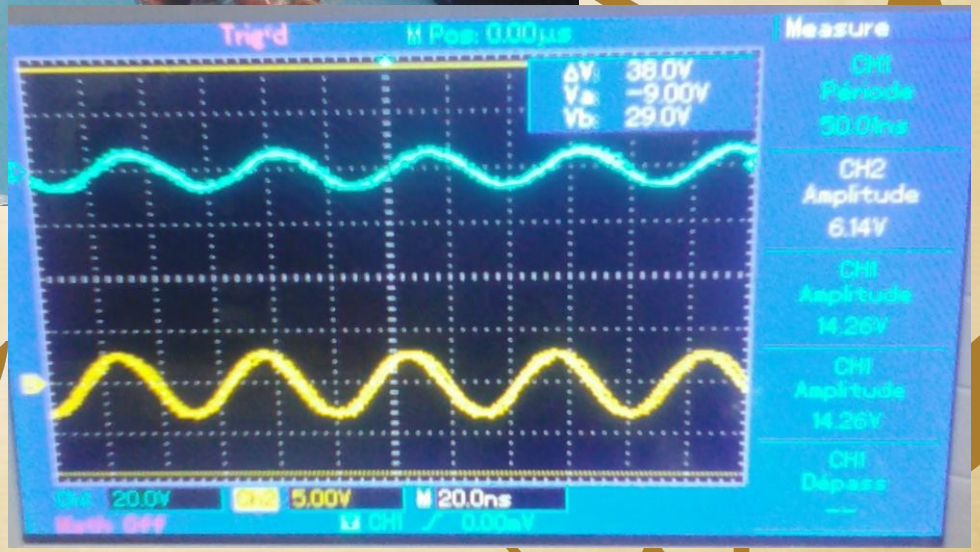





DQ8052


A SUCCESSFUL EXPERIENCE!

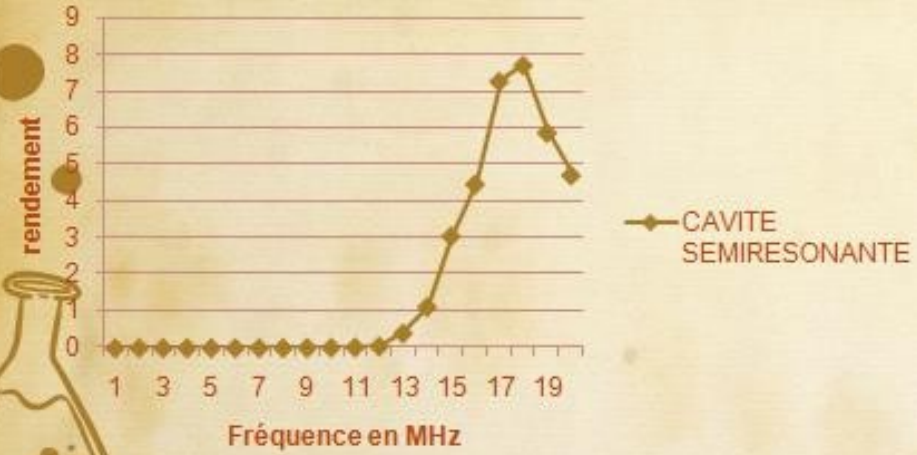



Trig'd
M Pos: 0.00µs
Measure
ΔV: 38.0V
Va: -9.00V
Vb: 29.0V
CH1
Période
50.0ns
CH2
Amplitude
6.14V
CH1
Amplitude
14.26V
CH1
Amplitude
14.26V
CH1
Dépass

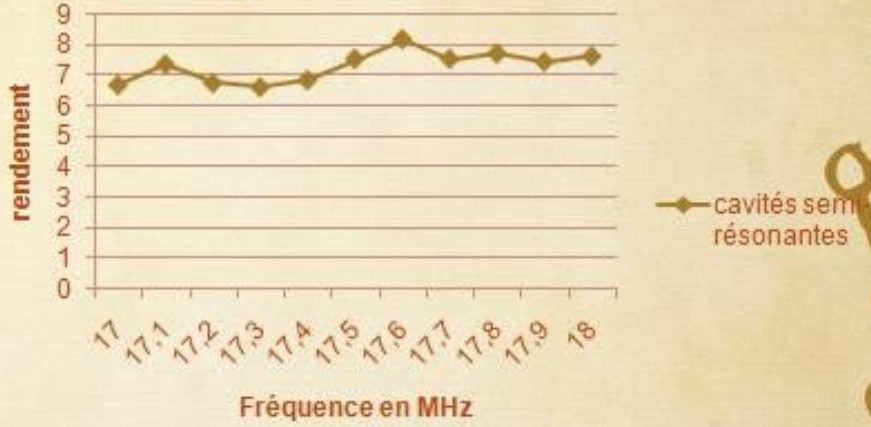
Ch1 20.0V
Ch2 5.00V
M 20.0ns
Math Off
CH1

À la recherche des valeurs exactes des fréquences de résonance

CAVITE SEMI-RESONANTE



cavités semi-résonantes

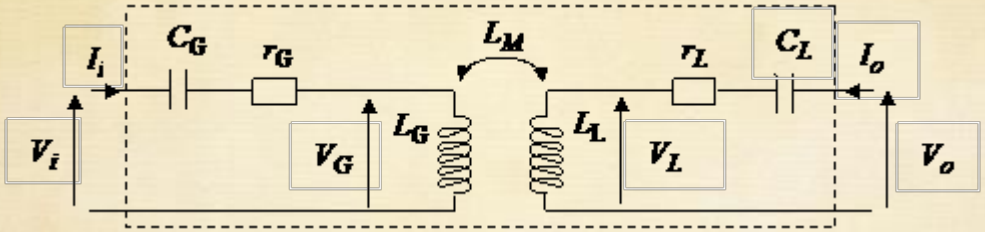


conclusion

- The semi-resonant cavity provides resonant coupling for highly reliable transfer of electrical power, and is resistant to misalignment and distance between transmitter and receiver.
- It can be further improved by silvering the wires and adding two resonators in proximity
- This can be adapted for cardiac implantation, insofar as the cavities can be designed to be worn by the patient.

APPENDIX 1

The most general electrical diagram concerning two resonant circuits coupled by the field associated with the inductances is as follows :



Most general equivalent diagram for the study of magnetic coupling between resonant circuits
Initially, the system can be considered in the form of a quadripole without taking into account the final load. To simplify the expressions, we use the following notations:

$$\begin{cases} x = \dfrac{1}{L_I C_I \omega^2} \\ y = \dfrac{1}{L_G C_G \omega^2} \end{cases} \quad \text{and} \quad \begin{cases} Q_G = \dfrac{L_G \omega}{r_G} \\ Q_I = \dfrac{L_I \omega}{r_I} \end{cases}$$

Note that it is implicitly assumed that losses are associated only with inductances. An elementary calculation leads to the following matrix form:

APPENDIX 1

$$\begin{pmatrix} V_i \\ V_o \end{pmatrix} = j\omega \begin{pmatrix} L_G(1-x+j/Q_G) & L_M \\ L_M & L_L(1-y+j/Q_L) \end{pmatrix} \begin{pmatrix} I_i \\ I_0 \end{pmatrix}$$

The determinant of this matrix is :

$$Det = j\omega L_G L_L \left[(1-x+j/Q_G)(1-y+j/Q_L) - k^2\right]$$

It is quite easy to show that this determinant never cancels out, so the matrix can always be inverted. Moreover, this matrix is very similar to the magnetic coupling matrix, differing only in the diagonal terms in brackets. It is therefore tempting to define a generalised coupling coefficient by the formula :

$$K^2 = \frac{L_M^2}{L_G(1-x+j/Q_G)L_L(1-y+j/Q_L)}$$

Using the definition of the magnetic coupling coefficient we obtain :

$$K^2 = \frac{k^2 Q_L Q_G}{\left[j+Q_L(1-x)\right]\left[j+Q_G(1-y)\right]}$$

If we pose : $\begin{cases} A_L = \dfrac{Q_L}{j+Q_L(1-x)} \\ A_G = \dfrac{Q_G}{j+Q_G(1-y)} \end{cases}$ We obtain : $K^2 = k^2 A_L A_G$

APPENDIX 1

The original coupling is therefore multiplied by a gain on each side of the link. Optimum values are obtained for :
$x = y = 1$
In other words, when the two circuits have the same resonance frequency and are tuned to it, we have :

$$|A_I|_{\max} = Q_I \qquad |A_G|_{\max} = Q_G$$

And finally :

$$|K^2|_{\max} = k^2 Q_I Q_G$$

Since the energy transferred is proportional to the square of the coupling coefficient, it is as if each resonance, that of the load and that of the generator, amplified the energy transfer by a factor equal to the quality factor of the corresponding circuit.

Source: www.tmms.co.jp/Nearfield/approche_pedagogique/Circuits_resonants_couples.htm

APPENDIX 2: PYTHON code N 2D circular turns

N spires circulaires 2D.py - C:\Doc Emna\tipe implantation cardiaque\N spires circulaires 2D.py

File Edit Format Run Options Windows Help

```
import numpy as np

# mu0 = 12.566370614e-7
mu0 = 4 * np.pi
I = 1.
N=int(input("donner le nombre de spires désirées"))

def circular_loop(R, C = [0., 0., 0.], N = 200):
    XP = R * np.cos(np.linspace(0, 2*np.pi, N)) - C[0]
    ZP = R * np.sin(np.linspace(0, 2*np.pi, N)) - C[1]
    YP = np.zeros(N) + C[2]
    return XP, YP, ZP

def biot_savart(xP, yP, zP, XM, YM, ZM):
    """
    xP, yP et zP sont des ndarrays à une dimension contenant
    les NP coordonnées des points P représentant la distribution
    de courant.

    XM, YM sont des ndarrays à deux dimensions obtenus typiquement
    par
    XM, YM = np.meshgrid(np.linspace(xmin, xmax, NX), np.linspace(ymin, ymax, NY))

    ZM est un ndarray à deux dimensions représentant la cote des points de la
    grille XM, YM (typiquement rempli de 0 : observation dans le plan z = 0).

    La fonction renvoit deux ndarrays de dimension (NX, NY) qui sont les
    composantes (utiles) du champ dans le plan d'observation.
    """

    P = np.array([xP, yP, zP])
    # P.shape = (3, NP)

    # Vecteur déplacement élémentaire le long de la distribution
    vec_dlP = P[:,1:] - P[:,:-1]
    # vec_dlP.shape = (3, NP-1)

    # Localisation du point courant de la distribution
    mid_dlP = (P[:,1:] + P[:,:-1]) / float(2)
    # mid_dlP.shape = (3, NP-1)
```

N spires circulaires 2D.py - C:\Doc Emna\tipe implantation cardiaque\N spires circulaires 2D.py

File Edit Format Run Options Windows Help

```
    XPM = XM[:,:,np.newaxis] - mid_dlP[0,:]
    YPM = YM[:,:,np.newaxis] - mid_dlP[1,:]
    ZPM = ZM[:,:,np.newaxis] - mid_dlP[2,:]
    # XPM.shape = YPM.shape = ZPM.shape = (NX, NY, NP)

    PMcubed = (XPM**2 + YPM**2 + ZPM**2)**(3/2.)
    # PMcubed.shape = (NX, NY, NP)

    # Éviter la division par zéro (trop proche des sources)
    PMcubed[PMcubed < 1e-6] = np.nan

    Xdl_cross_PM = - YPM * vec_dlP[2,:] + ZPM * vec_dlP[1,:]
    Ydl_cross_PM = - ZPM * vec_dlP[0,:] + XPM * vec_dlP[2,:]
    Zdl_cross_PM = - XPM * vec_dlP[1,:] + YPM * vec_dlP[0,:]
    # Xdl_cross_PM.shape = Ydl_cross_PM.shape = Zdl_cross_PM.shape =
    # (NX, NY, NP)

    Xdl_cross_PM_over_PMcubed = Xdl_cross_PM / PMcubed
    Ydl_cross_PM_over_PMcubed = Ydl_cross_PM / PMcubed
    Zdl_cross_PM_over_PMcubed = Zdl_cross_PM / PMcubed

    BX = np.sum(Xdl_cross_PM_over_PMcubed, axis = 2)
    BY = np.sum(Ydl_cross_PM_over_PMcubed, axis = 2)
    BZ = np.sum(Zdl_cross_PM_over_PMcubed, axis = 2)
    # BX.shape = BY.shape = BZ.shape = (NX, NY)

    BNorme = (BX**2 + BY**2 + BZ**2)**(.5)
    # BNorme.shape = (NX, NY)

    return mu0 * I * BX/(4 * np.pi), mu0 * I * BY/(4 * np.pi),\
           mu0 * I * BNorme/(4 * np.pi)

def Bzloop(z):
    return mu0 * I * R**2/(2 * (R**2 + z**2)**(3/2.))

import matplotlib.pyplot as plt
import matplotlib.cm as cm

R = 1

xP, yP, zP = circular_loop(R, [0., 0., 0.])
```

```
# Grille de NX*NY points
NX = 30
NY = 30

# Coordonnées min et max des points de la grille
xmax = 30
xmin = -xmax
ymax = 30
ymin = -ymax

# Les coordonnées des points de la grille sont répartis uniformément
# sur [xmin, xmax]x[ymin, ymax]
xM = np.linspace(xmin, xmax, NX)
yM = np.linspace(ymin, ymax, NY)

# Création de la grille
XM, YM = np.meshgrid(xM, yM)

# On impose la cote de la grille : z = 0
ZM = np.zeros((yM.size, xM.size))

# Calcul du champ magnétique
# La norme de B (BNorme) permet de normer le champ
BX, BY, BNorme = biot_savart(xP, yP, zP, XM, YM, ZM)



for i in range(1,N-1):
    xP, yP, zP = circular_loop(1+0.7*i, [0., 0., 0.])
    BX2, BY2, BNorme2 = biot_savart(xP, yP, zP, XM, YM, ZM)

    BX += BX2
    BY += BY2
    BNorme += BNorme2

plt.quiver(XM, YM, BX, BY, pivot = 'middle', units = 'width')

plt.imshow(BNorme, interpolation = 'bilinear', origin = 'lower',
                   cmap = cm.jet, extent = (xmin, xmax, ymin, ymax))
# plt.contour(XM, YM, BNorme, 10)
plt.show()
```

APPENDIX 3: PYTHON code N 2D polygonal turns

N spires polygone régulier 2D.py - C:\Doc Emna\tipe implantation cardiaque\N spires polygone

File Edit Format Run Options Windows Help

```
import numpy as np

# mu0 = 12.566370614e-7
mu0 = 4 * np.pi
PII=np.pi
I = 1.

M=int(input('donnez le nombre de cotes désiré pour votre spire'))

C=[0.,0.,0.]
N=200
def polygone_loop (RAY, C):
    XP = []
    ZP = []
    YP = []
    Xs = []
    Zs = []
    Ys = []
    for k in range (M):
        Xs.append(np.cos(2*k*PII/M)*RAY+C[0])
        Zs.append(np.sin(2*k*PII/M)*RAY+C[1])
        Ys.append(0+C[2])
    for i in range(M-1):
        X=np.linspace(Xs[i],Xs[i+1],N)
        XL=list(X)
        XP=XP+XL
        Y=np.zeros(N)
        YL=list(Y)
        YP=YP+YL
        Z=np.linspace(Zs[i],Zs[i+1],N)
        ZL=list(Z)
        ZP=ZP+ZL
    X=np.linspace(Xs[M-1],Xs[0],N)
    XL=list(X)
    XP=XP+XL
    Y=np.zeros(N)
    YL=list(Y)
    YP=YP+YL
    Z=np.linspace(Zs[M-1],Zs[0],N)
    ZL=list(Z)
    ZP=ZP+ZL
```

N spires polygone régulier 2D.py - C:\Doc Emna\tipe implantation cardiaque\N spires polygone régulier 2D.py

File Edit Format Run Options Windows Help

```
    ZP=ZP+ZL
    return XP, YP, ZP

def biot_savart(xP, yP, zP, XM, YM, ZM):
    """
    xP, yP et zP sont des ndarrays à une dimension contenant
    les NP coordonnées des points P représentant la distribution
    de courant.

    XM, YM sont des ndarrays à deux dimensions obtenus typiquement
    par
    XM, YM = np.meshgrid(np.linspace(xmin, xmax, NX), np.linspace(ymin, ymax, NY))

    ZM est un ndarray à deux dimensions représentant la cote des points de la
    grille XM, YM (typiquement rempli de 0 : observation dans le plan z = 0).

    La fonction renvoit deux ndarrays de dimension (NX, NY) qui sont les
    composantes (utiles) du champ dans le plan d'observation.
    """

    P = np.array([xP, yP, zP])
    # P.shape = (3, NP)

    # Vecteur déplacement élémentaire le long de la distribution
    vec_dlP = P[:,1:] - P[:,:-1]
    # vec_dlP.shape = (3, NP-1)

    # Localisation du point courant de la distribution
    mid_dlP = (P[:,1:] + P[:,:-1]) / float(2)
    # mid_dlP.shape = (3, NP-1)

    XPM = XM[:,:,np.newaxis] - mid_dlP[0,:]
    YPM = YM[:,:,np.newaxis] - mid_dlP[1,:]
    ZPM = ZM[:,:,np.newaxis] - mid_dlP[2,:]
    # XPM.shape = YPM.shape = ZPM.shape = (NX, NY, NP)

    PMcubed = (XPM**2 + YPM**2 + ZPM**2)**(3/2.)
    # PMcubed.shape = (NX, NY, NP)

    # Éviter la division par zéro (trop proche des sources)
    PMcubed[PMcubed < 1e-6] = np.nan
```

N spires polygone régulier 2D.py - C:\Doc Emna\tipe implantation cardiaque\N spires polygone régu

File Edit Format Run Options Windows Help

```
    Xdl_cross_PM = - YPM * vec_dlP[2,:] + ZPM * vec_dlP[1,:]
    Ydl_cross_PM = - ZPM * vec_dlP[0,:] + XPM * vec_dlP[2,:]
    Zdl_cross_PM = - XPM * vec_dlP[1,:] + YPM * vec_dlP[0,:]
    # Xdl_cross_PM.shape = Ydl_cross_PM.shape = Zdl_cross_PM.shape =
    # (NX, NY, NP)

    Xdl_cross_PM_over_PMcubed = Xdl_cross_PM / PMcubed
    Ydl_cross_PM_over_PMcubed = Ydl_cross_PM / PMcubed
    Zdl_cross_PM_over_PMcubed = Zdl_cross_PM / PMcubed

    BX = np.sum(Xdl_cross_PM_over_PMcubed, axis = 2)
    BY = np.sum(Ydl_cross_PM_over_PMcubed, axis = 2)
    BZ = np.sum(Zdl_cross_PM_over_PMcubed, axis = 2)
    # BX.shape = BY.shape = BZ.shape = (NX, NY)

    BNorme = (BX**2 + BY**2 + BZ**2)**(.5)
    # BNorme.shape = (NX, NY)

    return mu0 * I * BX/(4 * np.pi), mu0 * I * BY/(4 * np.pi),\
           mu0 * I * BNorme/(4 * np.pi)


def Bzloop(z):
    return mu0 * I * R**2/(2 * (R**2 + z**2)**(3/2.))

import matplotlib.pyplot as plt
import matplotlib.cm as cm

R=1

xP, yP, zP = polygone_loop(R, [0., 0., 0.])

# Grille de NX*NY points
NX = 30
NY = 30

L=int(input("donner le nombre de spires désirées"))

# Coordonnées min et max des points de la grille
xmax = 10
xmin = -xmax
```

```
ymax = 10
ymin = -ymax

# Les coordonnées des points de la grille sont répartis uniformément
# sur [xmin, xmax]x[ymin, ymax]
xM = np.linspace(xmin, xmax, NX)
yM = np.linspace(ymin, ymax, NY)

# Création de la grille
XM, YM = np.meshgrid(xM, yM)

# On impose la cote de la grille : z = 0
ZM = np.zeros((yM.size, xM.size))

BX, BY, BNorme = biot_savart(xP, yP, zP, XM, YM, ZM)


for i in range(1,L):
    xP, yP, zP = polygone_loop(R+i*0.1, [0., 0., 0.])
    BX2, BY2, BNorme2 = biot_savart(xP, yP, zP, XM, YM, ZM)

    BX += BX2
    BY += BY2
    BNorme += BNorme2
plt.quiver(XM, YM, BX, BY, pivot = 'middle', units = 'width')
plt.imshow(BNorme, interpolation = 'bilinear', origin = 'lower',
                   cmap = cm.jet, extent = (xmin, xmax, ymin, ymax))
plt.show()
```

APPENDIX 4: PYTHON N 3D circular turns code

N spires circulaires.py.py - C:\Doc Emna\tipe implantation cardiaque\N spires circulaires.py.py

File Edit Format Run Options Windows Help

```
import numpy as np

# mu0 = 12.566370614e-7
mu0 = 4 * np.pi
I = 1.

def circular_loop(R = 1e-2, C = [0., 0., 0.], N = 200):
    XP = R * np.cos(np.linspace(0, 2*np.pi, N)) - C[0]
    ZP = R * np.sin(np.linspace(0, 2*np.pi, N)) - C[1]
    YP = np.zeros(N) + C[2]
    return XP, YP, ZP

def biot_savart(xP, yP, zP, XM, YM, ZM):
    """
    xP, yP et zP sont des ndarrays à une dimension contenant
    les NP coordonnées des points P représentant la distribution
    de courant.

    XM, YM sont des ndarrays à deux dimensions obtenus typiquement
    par
    XM, YM = np.meshgrid(np.linspace(xmin, xmax, NX), np.linspace(ymin, ymax, NY))

    ZM est un ndarray à deux dimensions représentant la cote des points de la
    grille XM, YM (typiquement rempli de 0 : observation dans le plan z = 0).

    La fonction renvoit deux ndarrays de dimension (NX, NY) qui sont les
    composantes (utiles) du champ dans le plan d'observation.
    """

    P = np.array([xP, yP, zP])
    # P.shape = (3, NP)

    # Vecteur déplacement élémentaire le long de la distribution
    vec_dlP = P[:,1:] - P[:,:-1]
    # vec_dlP.shape = (3, NP-1)

    # Localisation du point courant de la distribution
    mid_dlP = (P[:,1:] + P[:,:-1]) / float(2)
    # mid_dlP.shape = (3, NP-1)
```

N spires circulaires.py.py - C:\Doc Emna\tipe implantation cardiaque\N spires circulaires.py.py

File Edit Format Run Options Windows Help

```
    XPM = XM[:,:,np.newaxis] - mid_dlP[0,:]
    YPM = YM[:,:,np.newaxis] - mid_dlP[1,:]
    ZPM = ZM[:,:,np.newaxis] - mid_dlP[2,:]
    # XPM.shape = YPM.shape = ZPM.shape = (NX, NY, NP)

    PMcubed = (XPM**2 + YPM**2 + ZPM**2)**(3/2.)
    # PMcubed.shape = (NX, NY, NP)

    # Éviter la division par zéro (trop proche des sources)
    PMcubed[PMcubed < 1e-6] = np.nan

    Xdl_cross_PM = - YPM * vec_dlP[2,:] + ZPM * vec_dlP[1,:]
    Ydl_cross_PM = - ZPM * vec_dlP[0,:] + XPM * vec_dlP[2,:]
    Zdl_cross_PM = - XPM * vec_dlP[1,:] + YPM * vec_dlP[0,:]
    # Xdl_cross_PM.shape = Ydl_cross_PM.shape = Zdl_cross_PM.shape =
    # (NX, NY, NP)

    Xdl_cross_PM_over_PMcubed = Xdl_cross_PM / PMcubed
    Ydl_cross_PM_over_PMcubed = Ydl_cross_PM / PMcubed
    Zdl_cross_PM_over_PMcubed = Zdl_cross_PM / PMcubed

    BX = np.sum(Xdl_cross_PM_over_PMcubed, axis = 2)
    BY = np.sum(Ydl_cross_PM_over_PMcubed, axis = 2)
    BZ = np.sum(Zdl_cross_PM_over_PMcubed, axis = 2)
    # BX.shape = BY.shape = BZ.shape = (NX, NY)

    BNorme = (BX**2 + BY**2 + BZ**2)**(.5)
    # BNorme.shape = (NX, NY)

    return mu0 * I * BX/(4 * np.pi), mu0 * I * BY/(4 * np.pi),\
           mu0 * I * BNorme/(4 * np.pi)

def Bzloop(z):
    return mu0 * I * R**2/(2 * (R**2 + z**2)**(3/2.))

import matplotlib.pyplot as plt
import matplotlib.cm as cm

R = float(input('donner le rayon de la spire désiré'))

xP, yP, zP = circular_loop(R, [0., 0., 0.])
```

N spires circulaires.py.py - C:\Doc Emna\tipe implantation cardiaque\N spires circulaires.py.py

File Edit Format Run Options Windows Help

```
# Grille de NX*NY points
NX = 30
NY = 30

# Coordonnées min et max des points de la grille
xmax = 10
xmin = -xmax
ymax = 10
ymin = -ymax

# Les coordonnées des points de la grille sont répartis uniformément
# sur [xmin, xmax]x[ymin, ymax]
xM = np.linspace(xmin, xmax, NX)
yM = np.linspace(ymin, ymax, NY)

# Création de la grille
XM, YM = np.meshgrid(xM, yM)

# On impose la cote de la grille : z = 0
ZM = np.zeros((yM.size, xM.size))

# Calcul du champ magnétique
# La norme de B (BNorme) permet de normer le champ
BX, BY, BNorme = biot_savart(xP, yP, zP, XM, YM, ZM)

L=int(input("donner le nombre de spires désirées"))
ZP=[i*0.1 for i in range(1, L//2)]
ZN=[i*(-0.1) for i in range (1,L//2)]
zspires=ZP+ZN

for z in zspires:
    xP, yP, zP = circular_loop(R, [0., 0., z])
    BX2, BY2, BNorme2 = biot_savart(xP, yP, zP, XM, YM, ZM)

    BX += BX2
    BY += BY2
    BNorme += BNorme2
plt.quiver(XM, YM, BX, BY, pivot = 'middle', units = 'width')
plt.imshow(BNorme, interpolation = 'bilinear', origin = 'lower',
                   cmap = cm.jet, extent = (xmin, xmax, ymin, ymax))
plt.show()
```

APPENDIX 4: PYTHON code N 3D polygonal turns

*N spires polygone régulier.py - C:\Doc Emna\tipe implantation cardiaque\N spires polygone r

File Edit Format Run Options Windows Help

```
import numpy as np

# mu0 = 12.566370614e-7
mu0 = 4 * np.pi
PII=np.pi
I = 1.

M=int(input('donnez le nombre de cotes désiré pour votre spire'))

C=[0.,0.,0.]
N=200
def polygone_loop (RAY, C):
    XP = []
    ZP = []
    YP = []
    Xs = []
    Zs = []
    Ys = []
    for k in range (M):
        Xs.append(np.cos(2*k*PII/M)*RAY-C[0])
        Zs.append(np.sin(2*k*PII/M)*RAY-C[1])
    for i in range(M-1):
        X=np.linspace(Xs[i],Xs[i+1],N)
        XL=list(X)
        XP=XP+XL
        Z=np.linspace(Zs[i],Zs[i+1],N)
        ZL=list(Z)
        ZP=ZP+ZL

    X=np.linspace(Xs[M-1],Xs[0],N)
    XL=list(X)
    XP=XP+XL
    YP=[C[2]]*(N*M)
    Z=np.linspace(Zs[M-1],Zs[0],N)
    ZL=list(Z)
    ZP=ZP+ZL
    return XP, YP, ZP
```

N spires polygone régulier.py - C:\Doc Emna\tipe implantation cardiaque\N spires polygone régulier.py

File Edit Format Run Options Windows Help

```
def biot_savart(xP, yP, zP, XM, YM, ZM):
    """
    xP, yP et zP sont des ndarrays à une dimension contenant
    les NP coordonnées des points P représentant la distribution
    de courant.

    XM, YM sont des ndarrays à deux dimensions obtenus typiquement
    par
    XM, YM = np.meshgrid(np.linspace(xmin, xmax, NX), np.linspace(ymin, ymax, NY))

    ZM est un ndarray à deux dimensions représentant la cote des points de la
    grille XM, YM (typiquement rempli de 0 : observation dans le plan z = 0).

    La fonction renvoit deux ndarrays de dimension (NX, NY) qui sont les
    composantes (utiles) du champ dans le plan d'observation.
    """

    P = np.array([xP, yP, zP])
    # P.shape = (3, NP)

    # Vecteur déplacement élémentaire le long de la distribution
    vec_dlP = P[:,1:] - P[:,:-1]
    # vec_dlP.shape = (3, NP-1)

    # Localisation du point courant de la distribution
    mid_dlP = (P[:,1:] + P[:,:-1]) / float(2)
    # mid_dlP.shape = (3, NP-1)

    XPM = XM[:,:,np.newaxis] - mid_dlP[0,:]
    YPM = YM[:,:,np.newaxis] - mid_dlP[1,:]
    ZPM = ZM[:,:,np.newaxis] - mid_dlP[2,:]
    # XPM.shape = YPM.shape = ZPM.shape = (NX, NY, NP)

    PMcubed = (XPM**2 + YPM**2 + ZPM**2)**(3/2.)
    # PMcubed.shape = (NX, NY, NP)

    # Éviter la division par zéro (trop proche des sources)
    PMcubed[PMcubed < 1e-6] = np.nan

    Xdl_cross_PM = - YPM * vec_dlP[2,:] + ZPM * vec_dlP[1,:]
    Ydl_cross_PM = - ZPM * vec_dlP[0,:] + XPM * vec_dlP[2,:]
```

*N spires polygone régulier.py - C:\Doc Emna\tipe implantation cardiaque\N spires polygone régulier

File Edit Format Run Options Windows Help

```
    Xdl_cross_PM = - YPM * vec_dlP[2,:] + ZPM * vec_dlP[1,:]
    Ydl_cross_PM = - ZPM * vec_dlP[0,:] + XPM * vec_dlP[2,:]
    Zdl_cross_PM = - XPM * vec_dlP[1,:] + YPM * vec_dlP[0,:]
    # Xdl_cross_PM.shape = Ydl_cross_PM.shape = Zdl_cross_PM.shape =
    # (NX, NY, NP)

    Xdl_cross_PM_over_PMcubed = Xdl_cross_PM / PMcubed
    Ydl_cross_PM_over_PMcubed = Ydl_cross_PM / PMcubed
    Zdl_cross_PM_over_PMcubed = Zdl_cross_PM / PMcubed

    BX = np.sum(Xdl_cross_PM_over_PMcubed, axis = 2)
    BY = np.sum(Ydl_cross_PM_over_PMcubed, axis = 2)
    BZ = np.sum(Zdl_cross_PM_over_PMcubed, axis = 2)
    # BX.shape = BY.shape = BZ.shape = (NX, NY)

    BNorme = (BX**2 + BY**2 + BZ**2)**(.5)
    # BNorme.shape = (NX, NY)

    return mu0 * I * BX/(4 * np.pi), mu0 * I * BY/(4 * np.pi),\
           mu0 * I * BNorme/(4 * np.pi)

def Bzloop(z):
    return mu0 * I * R**2/(2 * (R**2 + z**2)**(3/2.))

import matplotlib.pyplot as plt
import matplotlib.cm as cm

R=float(input('donner le rayon désiré du polygone='))

xP, yP, zP = polygone_loop(R, [0., 0., 0.])

# Grille de NX*NY points
NX = 30
NY = 30

L=int(input("donner le nombre de spires désirées"))

# Coordonnées min et max des points de la grille
# l'induction doit être sur un minimum de 7cm
xmax = 10
xmin = -xmax
```

```
ymax = 10
ymin = -ymax

# Les coordonnées des points de la grille sont répartis uniformément
# sur [xmin, xmax]x[ymin, ymax]
xM = np.linspace(xmin, xmax, NX)
yM = np.linspace(ymin, ymax, NY)

# Création de la grille
XM, YM = np.meshgrid(xM, yM)

# On impose la cote de la grille : z = 0
ZM = np.zeros((yM.size, xM.size))

BX, BY, BNorme = biot_savart(xP, yP, zP, XM, YM, ZM)

#on suppose que le fil est de diamètre 1mm

ZP=[i*0.1 for i in range(1, L//2)]
ZN=[i*(-0.1) for i in range (1,L//2)]
zspires=ZP+ZN

for z in zspires:
    xP, yP, zP = polygone_loop(R, [0.,0.,z])
    BX2, BY2, BNorme2 = biot_savart(xP, yP, zP, XM, YM, ZM)

    BX += BX2
    BY += BY2
    BNorme += BNorme2
plt.quiver(XM, YM, BX, BY, pivot = 'middle', units = 'width')
plt.imshow(BNorme, interpolation = 'bilinear', origin = 'lower',
                   cmap = cm.jet, extent = (xmin, xmax, ymin, ymax))
plt.show()
```