```
In [1]: %matplotlib inline
```

```
In [2]: import matplotlib.pyplot as plt
        from matplotlib import animation, colors, rc
        from mpl_toolkits.mplot3d import Axes3D
        import matplotlib.ticker as mtick
        from IPython.display import set_matplotlib_formats, HTML, Image

        set_matplotlib_formats('png', 'pdf')
        rc('animation', html='html5')

        import numpy as np
        import scipy.special as spec
        import scipy.integrate as inte
        import sympy as sp
        from sympy import vector
        sp.init_printing()
```

```
In [3]: # Variables mathématiques
        Cart = vector.CoordSysCartesian('R')

        x, y = sp.symbols('x y', real=True)
        r = sp.symbols('r', positive=True)
        omega = sp.symbols('omega', positive = True) # Pulsation
        t = sp.symbols('t', positive=True) # Temps
        Ic = sp.symbols('I', complex=True) # Courant électrique

        mu = sp.symbols("mu0", positive = True) # Perméabilité magnétique du milieu
        c = sp.symbols("c", positive=True)      # Célérité de la lumière dans le milieu
        k = omega/c       # relation de dispersion par défaut
```

1 Position du problème

Étant donné un fil électrique très long parcouru par un courant électrique i, on cherche à déterminer le champ magnétique créé par phénomène d'induction dans l'espace autour.

On peut le détecter en approchant une boussole, de la limaille de fer ou d'autres aimants permanents du fil, voire approcher un autre fil électrique lui aussi parcouru par un courant.

Le développement pour trouver l'expression du champ magnétique créé par un courant constant I est classique et mène à la solution $\mathbf{B}(\mathbf{r}) = \frac{\mu_0 I}{2\pi\|\mathbf{r}\|}\mathbf{e}_\theta$.

Ici, la question est celle d'un courant variable $i(t)$. On réussit à déterminer l'expression du champ créé lorsque i est sinusoïdal en utilisant des potentiels retardés.

Ce cahier Jupyter développe l'implémentation en `Python` d'objets et fonctions permettant de visualiser la propagation des ondes magnétiques crées par un courant variable, avec la possibilité d'exploiter plusieurs types de visualisations et des animations.

2 Implémentations en Python du champ magnétique et du courant électrique

Le courant et le champ magnétique seront représentés par des instances des classes `current` et `Field`, qui sont définies dans cette section.

Chaque courant, par exemple, sera un objet de type `Current`, dont les attributs, tels que `frequences`, `intensites`, `expr`, `func` contiendront les caractéristiques du courant, son expression mathématique, et une fonction numérique permettant de le calculer.

2.1 Courant électrique : classe `current`

La cellule suivante définit les courants électriques comme une classe Python `Current`.

```
In [4]: class Current:
            # Composante du courant électrique de pulsation omega
            cour_component = Ic*sp.exp(sp.I*omega*t)

            def __init__(self, intens=None, puls=None, phas=None):
                """
                Étant donné le spectre (intensités et pulsations), initialise le courant
                en attribuant les fréquences/pulsations du courant, les intensités
                (complexes) associées. L'argument d'une intensité complexe correspond
                au déphasage de la composante du courant associée.

                Si les phases sont précisées, elles sont ajoutées aux arguments des
                intensités."""
                if intens is not None:
                    if phas is not None:
                        intens = np.asarray(intens)*np.exp(1j*np.asarray(phas))
                    self._spectre(intens, puls)

            def _spectre(self, intens, puls):
                """
                Définit la fonction numérique du courant et son expression via
                ses données spectrales, stockées dans les attributs intensites,
                pulsations, et frequences.
                Ne pas utiliser directement."""
                self.intensites = intens
                self.pulsations = puls
                self.frequences = self.pulsations/(2*np.pi)

                spector = zip(intens,puls)
                cour = sum([self.cour_component.subs({Ic:i, omega:om}) \
                        for i,om in spector ])
                cour_re = sp.re(cour)

                cour_func = sp.lambdify((t),cour_re,modules=['numpy'])

                self.expr = cour_re
                self.func = cour_func

            def expression(self, expr):
                """
                Définit la fonction numérique du courant via son expression.
                Renvoie l'objet, donc on peut définir un courant par son
                expression en écrivant
                courant = Current().expression(expr)"""
                self.expr = expr
                self.func = sp.lambdify(t, expr, modules=['numpy'])
                return self

            def fft(self, fs, N):
                """
                Calcule le spectre du courant à partir de sa fonction numérique,
                définit les attributs intensites, pulsations et frequences
                À utiliser si l'objet a été défini sans passer de données spectrales.
                fs : Fréquence d'échantillonnage
                N : taille de l'échantillon"""
                sample_time = np.linspace(-N/fs,N/fs,N+1)
                samples = self.func(sample_time)
```

```python
        self.intensites = np.fft.rfft(samples) # Intensités
        self.pulsations = np.fft.rfftfreq(N, d=1/fs) # Pulsations associées
        self.frequences = self.pulsations/(2*np.pi)

    ## Méthodes de dessin
    def draw(self, tmin, tmax, N=1000, title=None):
        """
        Représentation graphique de la fonction i(t),
        stockée dans l'attribut 'self.graphe'

        tmin,tmax : intervalle de temps où tracer
        N : nombre de points (défaut 1000)
        custTitle : titre (facultatif)
        """
        times = np.linspace(tmin, tmax, N)

        fig,ax=plt.subplots(1,1,figsize=(8,5))
        ax.grid(True)

        if title:
            ax.set_title(title)
        else:
            ax.set_title(r"Courant électrique")
        ax.plot(times, self.func(times))
        ax.set_xlabel(r"Temps $t$ $(\mathrm{s})$")
        ax.set_ylabel(r"Intensité du courant $i$ ($A$)")
        fig.tight_layout()
        self.graphe = fig

    def drawfft(self):
        """
        Réprésentation graphique du spectre du courant,
        stockée dans l'attribut graphefft
        """
        fig, (ax0,ax1) = plt.subplots(2,1, figsize=(8,8))

        xlbl = r'Pulsation $\omega$ ($\mathrm{Hz}$)'

        ax0.grid(True)
        ax0.set_xlabel(xlbl)
        ax0.set_ylabel(r"Amplitude $I(\omega)$ ($\mathrm{A}$)")
        ax0.plot(self.pulsations, np.absolute(self.intensites))

        ax1.grid(True)
        ax1.set_xlabel(xlbl)
        ax1.set_ylabel(r"Phase $\phi(\omega)$ ($\mathrm{rad}$)")
        ax1.plot(self.pulsations, np.angle(self.intensites))

        ax0.set_title(r"Spectre en fréquence du courant $i(t)$")
        fig.tight_layout()

        self.graphefft = fig
```

Les objets de type `Current` représentent les courants électriques, avec des attributs et méthodes permettant de les exploiter.

Étant fournies les informations sur le spectre d'un courant (intensités des composantes harmoniques `intensites`, pulsations et phases de celles-ci `pulsations` et `phases`, représentées par des listes ou `ndarray` NumPy de mêmes dimensions), on définit sa représentation `courant` en écrivant:

```
courant = Current(intensites, pulsations, phases)
```

Mais il n'est pas nécessaire d'avoir ces données spectrales pour définir le courant (voir plus bas).

Les attributs utiles d'un objet de type `Current` sont: * les données spectrales (`intensites`, `pulsations`, `frequences`) * l'expression mathématique `expr`.

Les méthodes utiles sont: * `func`, la fonction numérique du temps $i(t)$ qui permet de calculer la valeur du courant à un instant t. Pour calculer la valeur du courant (représenté par l'objet `courant` ci-après) à l'instant $t = 1\,\mathrm{s}$, par exemple, on utilise

```
courant.func(1)
```

- `expression`, qui prend en argument l'expression et définit la fonction du temps $i(t)$ avec, et renvoie l'objet, ce qui permet d'écrire

  ```
  courant = Current().expression(expr)
  ```

  pour définir un courant par son expression en une ligne, au lieu de (plus long)

  ```
  courant = Current()
  courant.expression(expr)
  ```

- `fft` qui prend en arguments une fréquence d'échantillonnage `fs` et une taille d'échantillon `N` et calcule la transformée de Fourier rapide (FFT) du signal échantillonné avec `N` points sur l'intervalle $[-N/f_s, N/f_s]$.
- les méthodes de dessin, à savoir
  - `draw` qui prend en argument deux instants `t0` et `t1` entre lesquels dessiner $i(t)$, avec deux arguments facultatifs `N` et `title` qui correspondent au nombre de points pour le dessin (1000 par défaut), et le titre éventuel à donner (Courant électrique $i\,(\mathrm{A})$ par défaut)
  - `drawfft` qui ne prend pas d'arguments et dessine le spectre du courant (module $|I(\omega)|$ et phase $\varphi(\omega)$).

2.2 Domaine spatial : classe `Domain`

```
In [5]: class Domain:

            def __init__(self, xm, ym, J, x0=None, y0=None, eps=0):
                if x0 is None:
                    x0 = -xm
                if y0 is None:
                    y0 = -ym

                self.xm = xm
                self.xs = np.linspace(x0,xm,J)
                self.ym = ym
                self.ys = np.linspace(y0,ym,J)
                self.J = J

                rad = np.sqrt(self.xs**2 + self.ys**2)
                cond = rad > eps

                self.rad = rad[cond]
                self.grid = np.meshgrid(self.xs[cond],
                        self.ys[cond])

            def __call__(self):
                return self.grid
```

Les objets de type `Domain` permettent de représenter des rectangulaires de l'espace $[x_0, x_m] \times [y_0, y_m]$.

Pour définir un domaine en connaissant x_m et y_m, avec `I` et `J` points selon les directions x et y respectivement, on écrit:

```
Omega = Domain(xm, ym, J)
```

Par défaut, le domaine est symétrisé (on prend $x_0 = -y_m$ et $y_0 = -y_m$). Pour au contraire préciser x_0 et y_0, il suffit de passer des arguments supplémentaires au constructeur `Domain` :

```
Omega = Domain(xm, ym, J, x0, y0)
```

et si on veut n'en préciser qu'un seul des deux, écrire

```
Omega = Domain(xm, ym, J, x0 = ...) ou
Omega = Domain(xm, ym, J, y0 = ...)
```

en fonction du contexte, et remplacer les pointillés par les valeurs voulues.

Si on veut exclure un disque de rayon ε centré en l'origine, préciser la paramètre `eps = ...` dans l'appel à `Domain` de la même façon que pour `x0` et `y0` au-dessus.

2.3 Champ magnétique : classe `Field`

La cellule suivante définit les champs magnétiques comme une classe Python `Field`, dont les attributs sont notamment l'expression formelle du champ (`expr`), et la fonction numérique qui permet de calculer le champ en un point (`func`).

(Attention code long)

```
In [121]: class Field:

              def __init__(self, intens=None, puls=None, phas=None):
                  # Célérité des ondes ; à modifier en fonction du milieu
                  self.cel = 3e8
                  # Composante du potentiel vecteur associée à la pulsation omega
                  self.A_component = sp.I*mu*Ic/4 * sp.hankel2(0,k*r)*sp.exp(sp.I*omega*t)
                  # Composante du champ magnétique associée à la pulsation omega
                  self.B_component = - sp.diff(self.A_component, r).simplify()

                  # Conversion de fonctions formelles à fonctions numériques
                  self.impl_modules = ['numpy',
                                       {"hankel2":spec.hankel2,
                                        #"sqrt":lambda x: np.sqrt(x+0j)
                                       }]

                  if not(intens is None):
                      if not(phas is None):
                          intens = np.asarray(intens)*np.exp(1j*np.asarray(phas))
                      self.pulsations = puls
                      self.frequences = puls/(2*np.pi)

                      c0 = self.cel
                      mu0_v = 4e-7*np.pi

                      spectral_data = zip(intens, puls)

                      orthC = sum([
                          self.B_component.subs({Ic: cur, omega:om, c:c0, mu:mu0_v}) \
                          for (cur,om) in spectral_data if (cur!=0 and om!=0) ])

                      orthExpr = sp.re(orthC)

                      orthFunc = sp.lambdify((r, t), orthExpr,
                          modules = self.impl_modules)

                      self.orthExpr = orthExpr
```

```
        self.orthFunc = orthFunc

        # Fonctions en cartésien
        rxy = sp.sqrt(x**2+y**2)

        self.orthFuncCart = sp.lambdify((x, y, t), orthExpr.subs({r:rxy}),
            modules = self.impl_modules)
        self.fieldExpr = (-y*Cart.i + x*Cart.j)*orthExpr.subs({r:rxy})/rxy
        self.fieldFunc = sp.lambdify((x, y, t),
            self.fieldExpr.to_matrix(Cart),
            modules = self.impl_modules)

def legende(self,t):
    """Définit la légende à donner aux graphes à l'instant t"""
    return r'$t= {:.3e}$'.format(t) + r"$\ \mathrm{s}$"

def _setup_plot(self, Omega, title=None):
    radii = Omega.rad

    fig = plt.figure(figsize=(8,5))
    ax = plt.axes()
    ax.set_xlim((np.amin(radii), np.amax(radii)))
    ax.grid(True)
    ax.set_xlabel("Distance $r$ (m)")
    ax.set_ylabel("Valeur du champ (T)")

    if title:
        ax.set_title(title)
    else:
        ax.set_title(r'Champ magnétique ' + r'$\mathbf{B}$' \
                     + ' créé par un courant variable')

    ax.yaxis.set_major_formatter(mtick.FormatStrFormatter('%.2e'))
    return fig, ax

def profile(self, Omega, times, title = None):
    """
    Profil du champ magnétique aux instants de 'times'
    (fonction de la distance au fil)
    """
    func = self.orthFunc
    radii = Omega.rad
    fig,ax = self._setup_plot(Omega, title)

    if hasattr(times, '__iter__'):
        for ti in times:
            champ = func(radii, ti)
            ax.plot(radii, champ, label=self.legende(ti))
    else:
        champ = func(radii, times)
        ax.plot(radii, champ, label=self.legende(times))
    ax.legend(loc='best')

    self.graph = fig


def _setup_surface(self, Omega):
    grid, radii = Omega(), Omega.rad
```

```
        fig = plt.figure(2, figsize=(8,6))
        fig.suptitle(r"Champ magnétique $\mathbf{B}$")
        ax = fig.add_subplot(111, projection='3d')
        ax.grid(True)

        return grid,fig, ax

    def surfacePlot(self, Omega, t):
        """
        Portrait du champ magnétique à l'instant t (surface)
        """
        func = np.vectorize(self.orthFuncCart)
        radii = Omega.rad
        grid,fig,ax = self._setup_surface(Omega)

        normB = func(*Omega(), t)

        ax.plot_wireframe(*grid, normB)

        self.surf = fig

    def _animParams(self, t0, t1, animtime, fps):
        """
        Calcule les paramètres d'animation (nombre d'images,
        intervalles en temps réel et temps vidéo entre chaque image)"""
        N = int(np.ceil(fps*animtime))
        dt = (t1-t0)/N
        interval = 1000/fps
        return N, dt, interval

    def _window(self, valMin,valMax):
        """
        Renvoie un couple correspondant aux bornes d'un intervalle
        contenant valMin et valMax avec 20% de marge"""
        delta = np.absolute(valMax-valMin)
        return (valMin-0.2*delta,valMax+0.2*delta)

    def animate(self, Omega, t0, t1, animtime=10, title = None, fps=25):
        """
        Construit une animation du profil du champ
        entre les temps t0 et t1"""

        # Paramètres d'animation
        N,dt,interval = self._animParams(t0,t1,animtime,fps)

        func = self.orthFunc
        radii = Omega.rad
        grid = Omega()

        fig, ax = self._setup_plot(Omega, title)

        line, = ax.plot([], [], lw=2)
        time_text = ax.text(0.02, 0.95, '',
                            transform=ax.transAxes)

        record = [func(radii,t0+i*dt) for i in range(N)]
```

```python
        # Cadrage
        ymax = np.nanmax(np.asarray(record))
        ymin = np.nanmin(np.asarray(record))
        ax.set_ylim(self._window(ymin,ymax))

        def init():
            line.set_data([],[])
            time_text.set_text(self.legende(t0))
            return line,

        def update(i):
            ti = dt*i+t0
            champ = record[i]
            line.set_data(radii, champ)
            time_text.set_text(self.legende(ti))
            return line,
        self.radiiRecord = record
        anim = animation.FuncAnimation(fig, update, init_func=init,
                    frames=N, interval=interval, blit=True)
        self.anim = anim
        return self.anim

    def animate3D(self, Omega, t0, t1, animtime=10, fps=25):
        """
        t0,t1 -> intervalle [t0,t1]
        fps : nombre d'images par seconde à générer"""
        # Paramètres d'animation
        N,dt,interval = self._animParams(t0,t1,animtime,fps)
        func = np.vectorize(self.orthFuncCart)
        grid,fig,ax = self._setup_surface(Omega)

        # Enregistrement des valeurs du champ à afficher
        record = [func(*grid, t0+i*dt) for i in range(N)]
        self.recordSurface = record

        # Cadrage
        zmin = np.nanmin(np.asarray(record))
        zmax = np.nanmax(np.asarray(record))
        zlims = self._window(zmin,zmax)

        # Initialisation
        surf = ax.plot_wireframe(*grid, record[0])
        time_text = ax.text2D(0.5,1,self.legende(t0),
                    horizontalalignment='center',
                    transform=ax.transAxes)
        ax.set_zlim(zlims)

        def update(i):
            ax.clear()
            ax.set_zlim(zlims)
            ti = i*dt + t0
            champ = record[i]
            time_text = ax.text2D(0.5,1,self.legende(ti),
                    horizontalalignment='center',
                    transform=ax.transAxes)
            data = ax.plot_wireframe(*grid, champ)
            return data, time_text
```

```
        anim = animation.FuncAnimation(fig,update,
                        frames=N,interval=interval)
        self.surfaceAnim = anim
        return self.surfaceAnim
```

On définit un champ magnétique `champ` via les données spectrales du courant qui l'a créé :

```
champ = Field(intensites, pulsations)
```

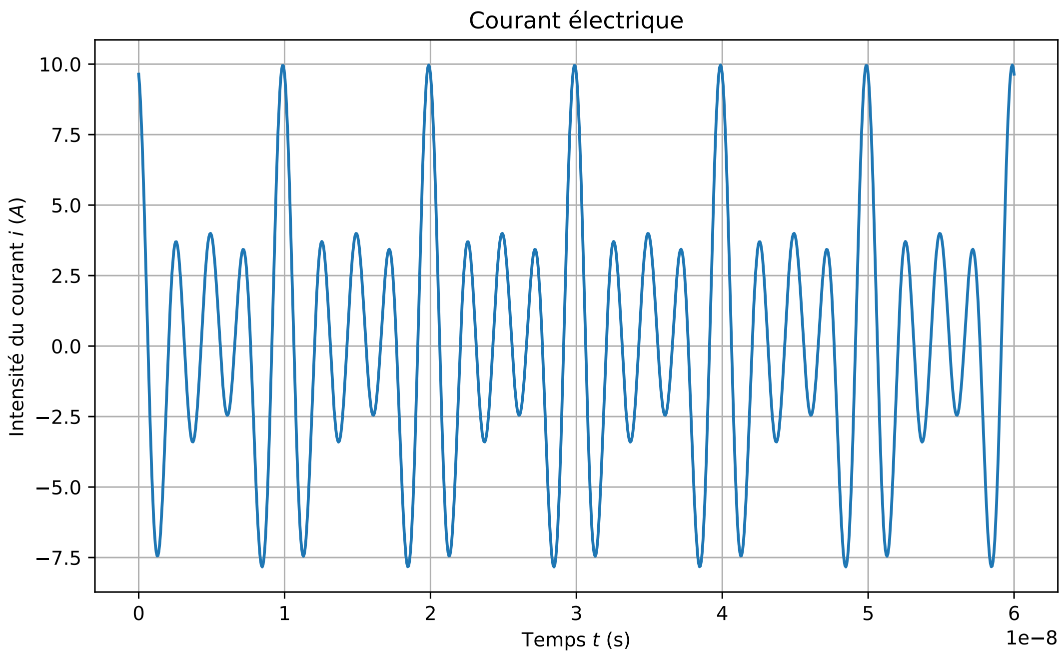
3 Exemples d'utilisation

3.1 Données initiales

Entrez dans la variable `freqs` les fréquences du courant voulu, et dans `phas` les phases. Exécutez la cellule (`Ctrl + Entrée` sur le clavier) pour définir la fonction de champ :

```
In [14]: k = omega/c # relation de dispersion

         freqs = np.array([n*1e8 for n in range(2,5)])

         puls = 2*np.pi*freqs # Pulsations associées
         intens = np.array([2,3,5]) # Intensités des composantes

         phases = np.array([0,0.3,0.3]) # Phases des composantes

         B_field = Field(intens, puls, phases)
```

```
In [15]: courant = Current(intens, puls, phases)
         courant.draw(0,6e-8)
```

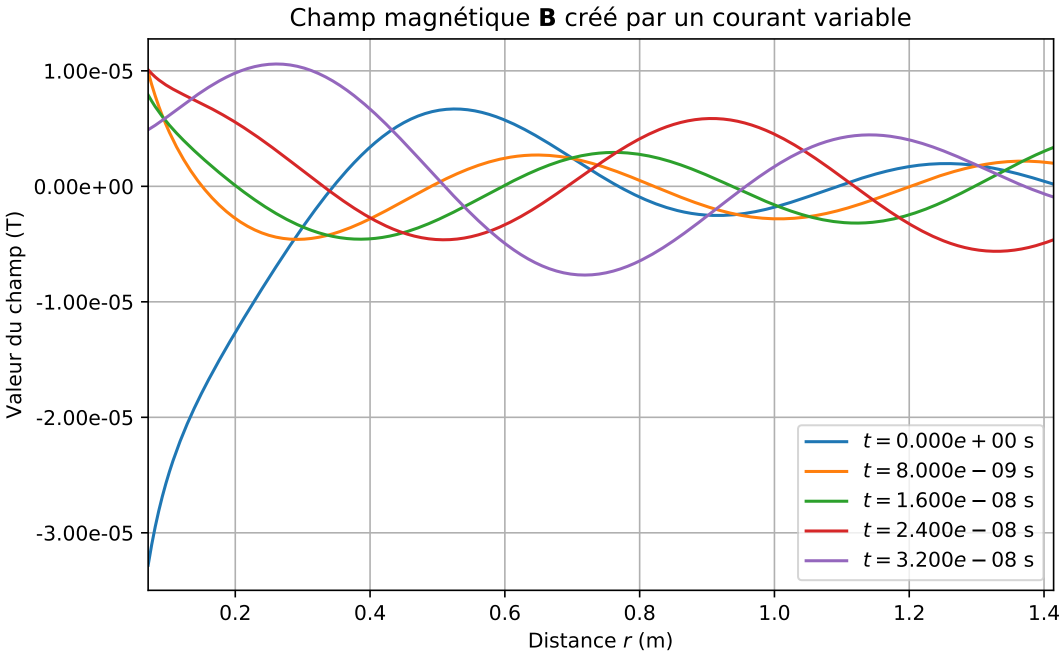


La cellule suivante définit les distances minimale et maximale pour lesquels tracer le profil du champ magnétique :

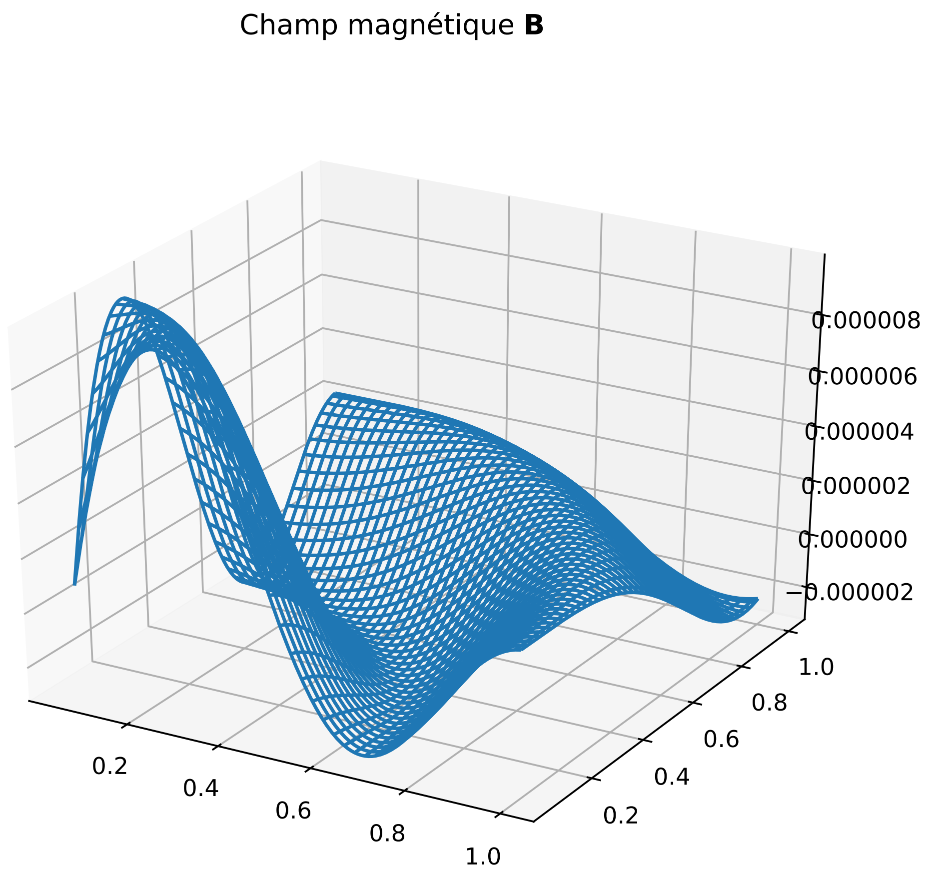
3.2 Profil du champ

3.2.1 Statique

In [16]:
```
Omega = Domain(1,1,256, 0.05, 0.05)

times = [1e-9*8*k for k in range(5)]

B_field.profile(Omega,times)
```



In [17]:
```
B_field.surfacePlot(Omega, 3.92e-8)
```

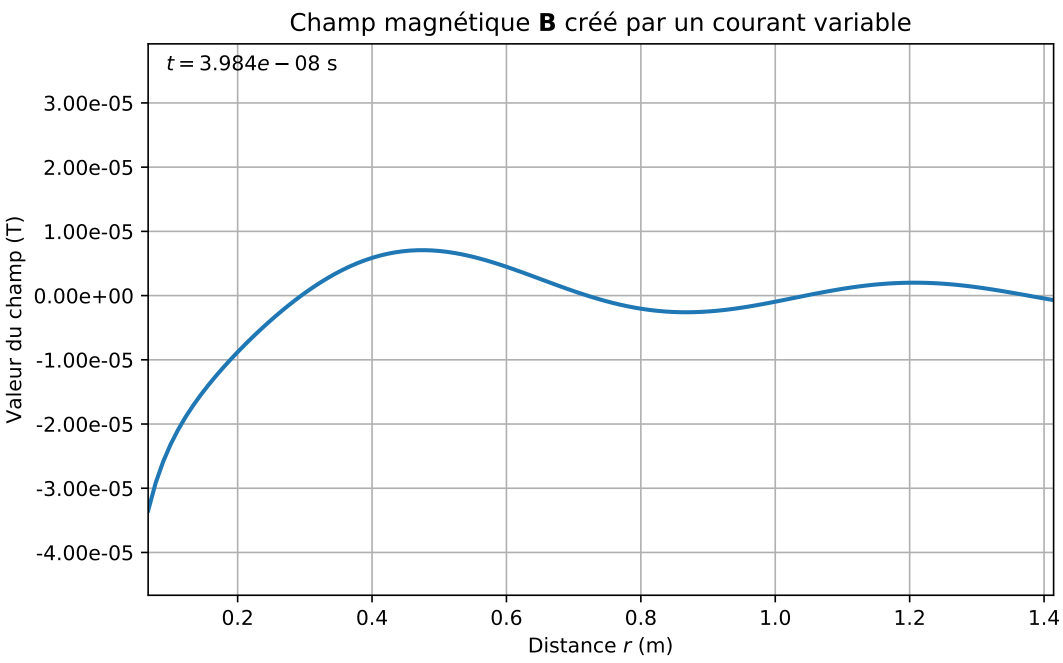


3.2.2 Animation

On peut animer le profil du champ magnétique entre deux instants t_0 et t_m :

In [19]:
```
Omega = Domain(1,1,128, 0,0,eps=0.06)
times_an = (0, 4e-8) # (t0, t1)

B_field.animate(Omega,*times_an)
```

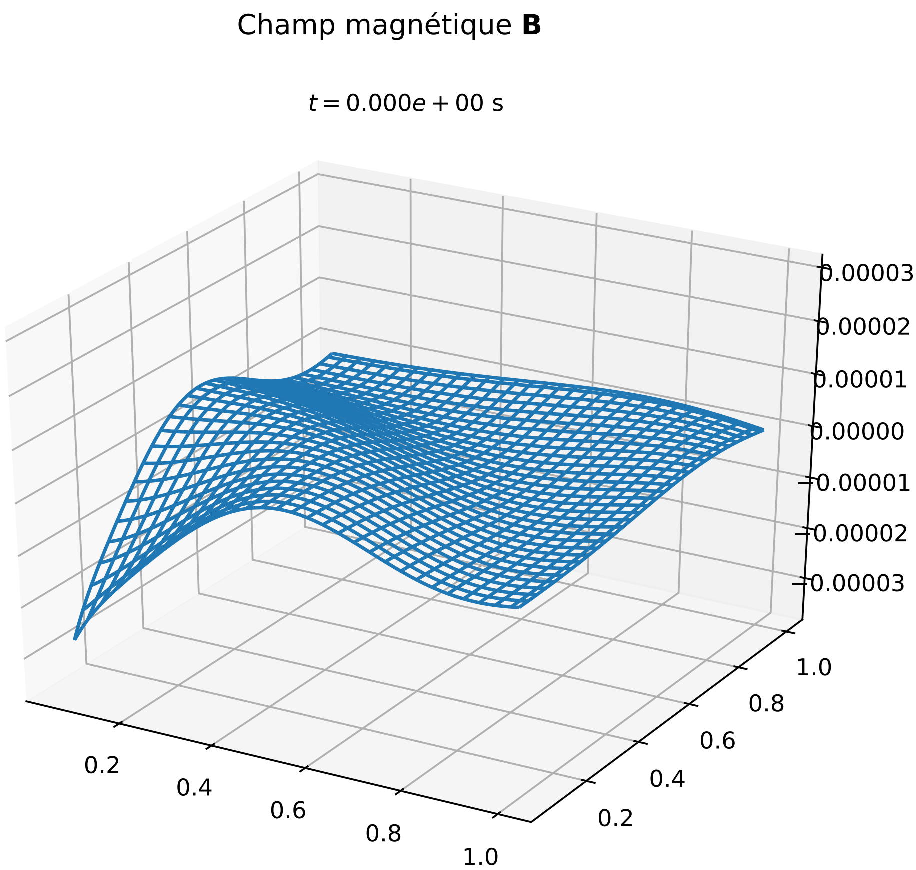
Out[19]:
```
<matplotlib.animation.FuncAnimation at 0x2ae2de5ef98>
```



La visualisation en tant que surface ondulante:

In [21]:
```
Omega = Domain(1,1,64,0,0,eps=0.08)
B_field.animate3D(Omega, *times_an)
```

Out[21]: <matplotlib.animation.FuncAnimation at 0x2ae2dd760b8>



4 Exemple d'application: Paquet d'ondes

On cherche à simuler le champ créé par des impulsions sinusoïdales de courant dans le fil électrique, de la forme

$$i(t) = I_0 \exp\left(-\frac{t^2}{2\tau^2}\right)\cos\left(\frac{t}{\tau}\right)$$

avec τ l'étendue de l'impulsion, de l'ordre de la dizaine de femtosecondes ($\approx 10^{-14}$ secondes).

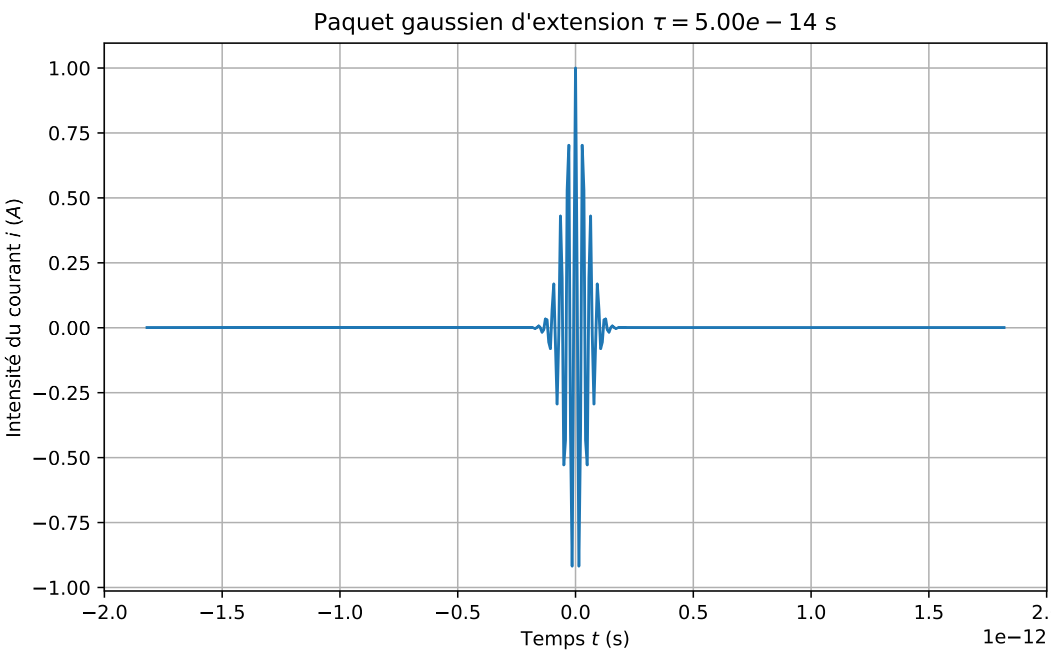
Pour définir un courant `Current` via son expression, il suffit d'initialiser un objet de classe `Current` en écrivant `out = Current()` par exemple, puis en utilisant la méthode `_expr` avec l'expression en argument.

```
In [23]: def gaussienne(tau):
             expr = sp.exp(-t**2/(2*tau**2))*sp.cos(10*t/tau)
             out = Current().expression(expr)
             return out
```

```
In [24]: courant = gaussienne(5e-14)
```

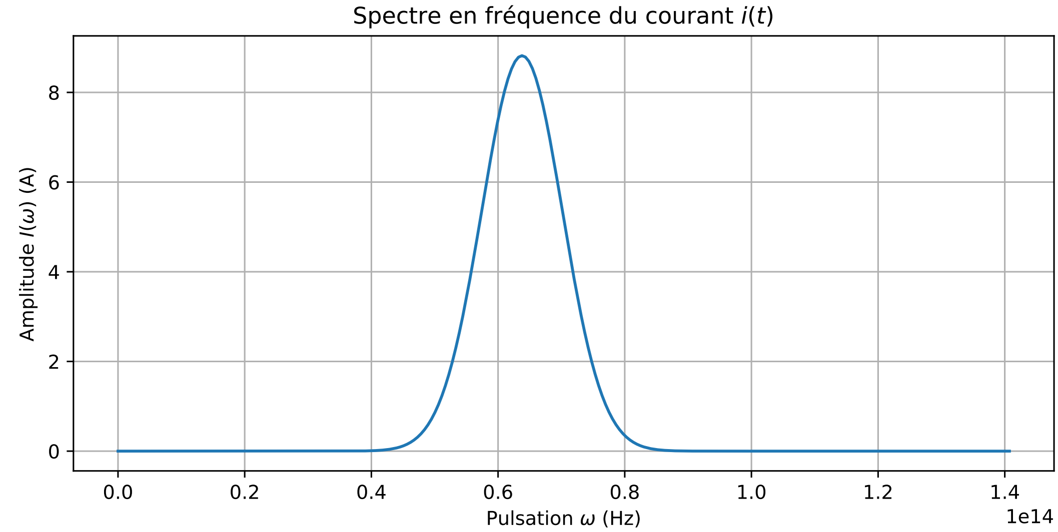
Courbe représentative du courant:

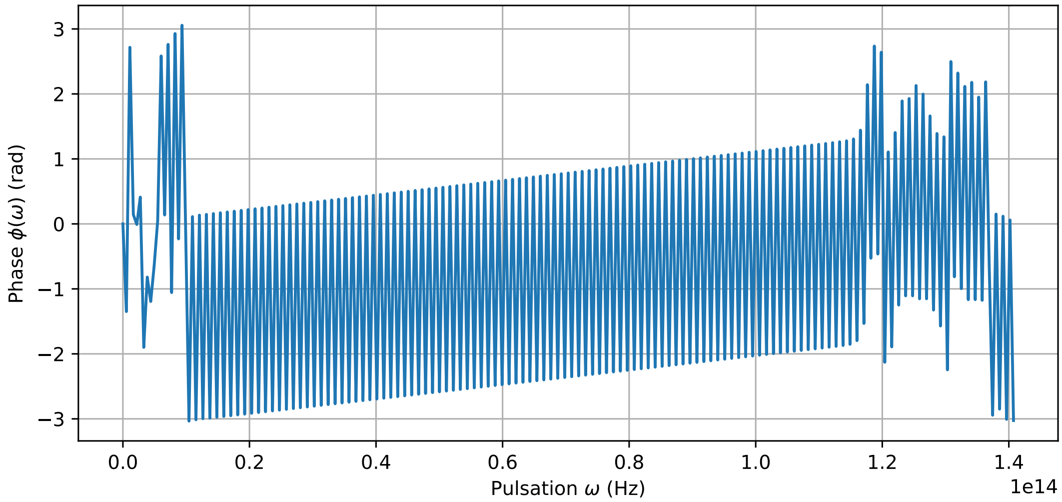
```
In [26]: N = 2**9 # Nombre d'échantillons
         fs = 2**48 # Fréquence d'échantillonnage

         titros = r"Paquet gaussien d'extension $\tau = {:.2e}$ s".format(5e-14)
         courant.draw(-N/fs,N/fs, N+1, titros)
```



Construction du spectre du courant via la méthode `bake_fft`:

```
In [27]: courant.fft(fs, N)
         courant.drawfft()
```

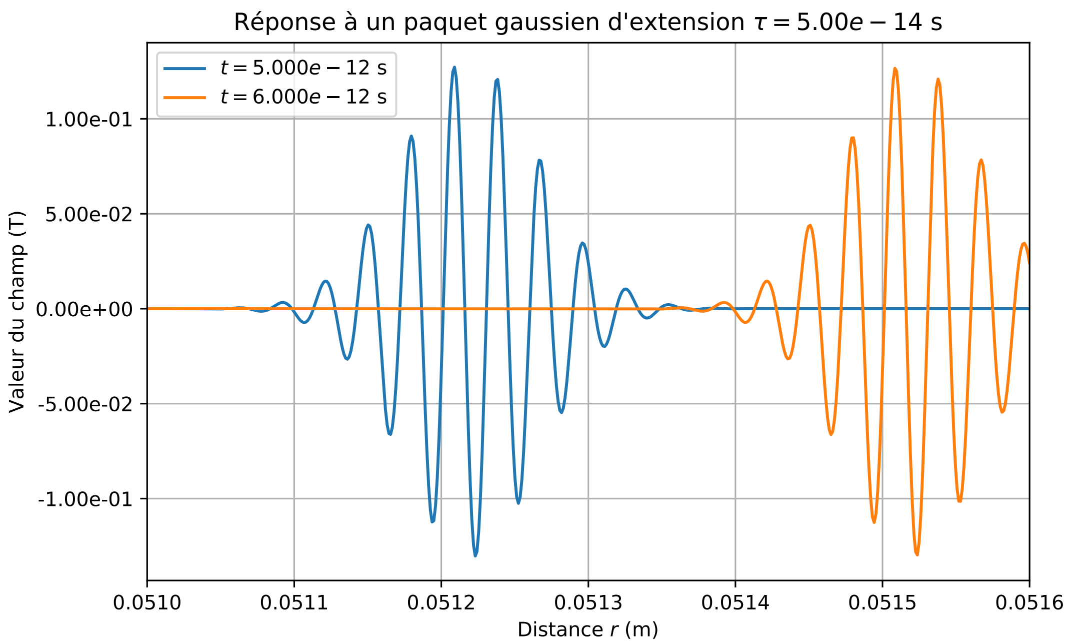





```
In [28]: k = omega/c

         reponseImpulsion = Field(courant.intensites,
                                  courant.pulsations)
```

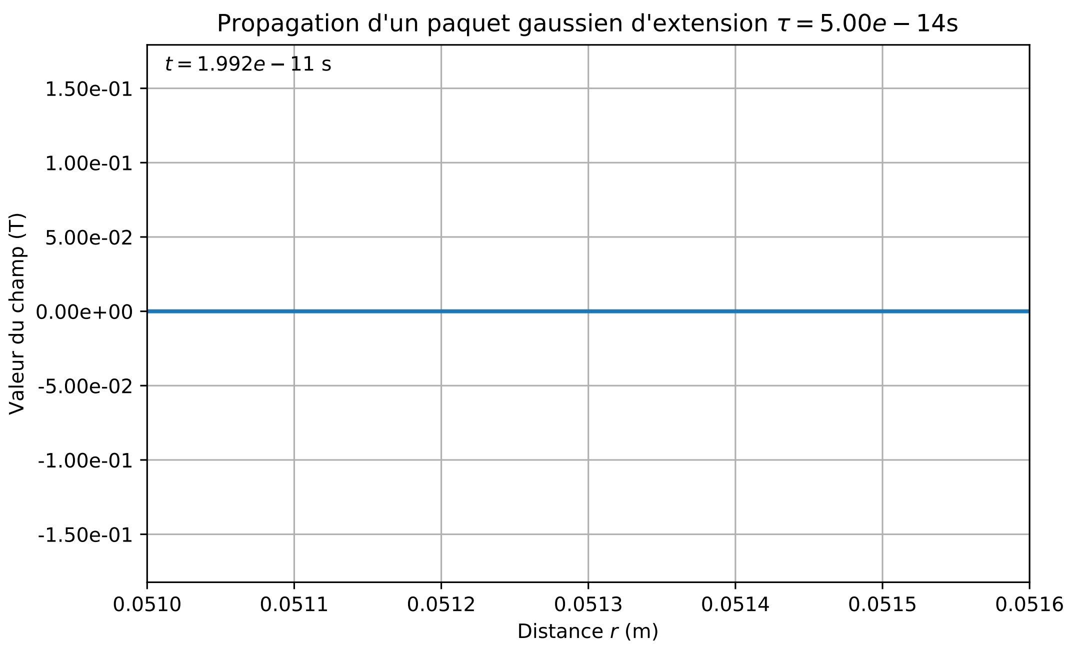
```
In [29]: rmin = 0.051
         rmax = 0.0516
         xmin = rmin/2**0.5
         xmax = rmax/2**0.5

         Omega = Domain(xmax,xmax,512,x0=xmin,y0=xmin)
```

```
In [31]: times = [1e-12*k for k in [5,6]]
         titre_gauss = "Réponse à un paquet gaussien d'extension " + \
                 r"$\tau={:.2e}$".format(5e-14) + " $\mathrm{s}$"
         reponseImpulsion.profile(Omega, times, title=titre_gauss)
```




```
In [33]: Omega = Domain(xmax,xmax,256,x0=xmin,y0=xmin)
         temps = (0,2e-11)
         titre = "Propagation d'un paquet gaussien d'extension " + \
                 r"$\tau = {:.2e}$".format(5e-14) + \
                 "$\mathrm{s}$"
         reponseImpulsion.animate(Omega,*temps, title=titre)
```

```
Out[33]: <matplotlib.animation.FuncAnimation at 0x2ae322789b0>
```



5 En milieu dispersif

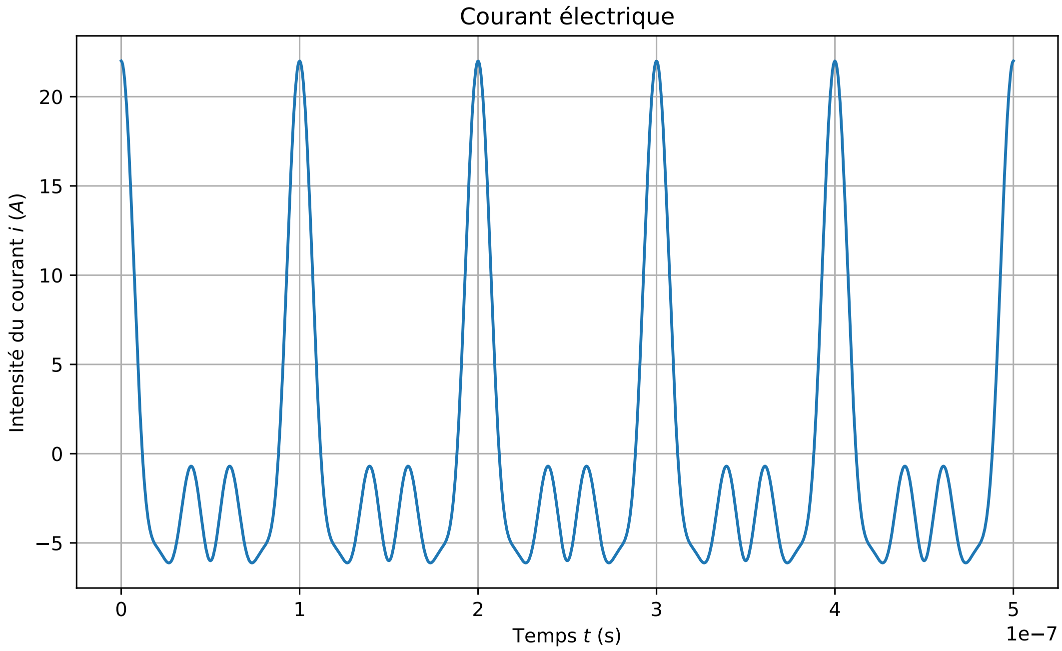
On s'intéresse à la propagation dans un plasma. La relation de dispersion (entre vecteur d'onde k et pulsation ω) dans un plasma de fréquence ω_0 est

$$k^2 = \frac{\omega^2 - \omega_0^2}{c^2}$$

Pour la démonstration, on utilisera le courant électrique suivant:

In [34]:
```
freqs = np.array([n*1e7 for n in range(1,6)])
puls = 2*np.pi*freqs
intens = np.array([7,7,5,1,2])

courant = Current(intens, puls)
```

In [35]:
```
courant.draw(0,5e-7)
```



On s'intéresse ici au champ induit par un courant à l'intérieur d'une gaine, qui peut être assimilée au vide, et de rayon 4 m, plongée dans un plasma.

In [113]:
```
omega0 = 2*np.pi*2.2e7
kplas = sp.Piecewise((sp.sqrt(omega**2-omega0**2)/c, omega >= omega0),
            (-sp.I/c*sp.sqrt(omega0**2-omega**2), True))
k = sp.Piecewise((kplas, r>4), (omega/c, True))
k
```

Out[113]:

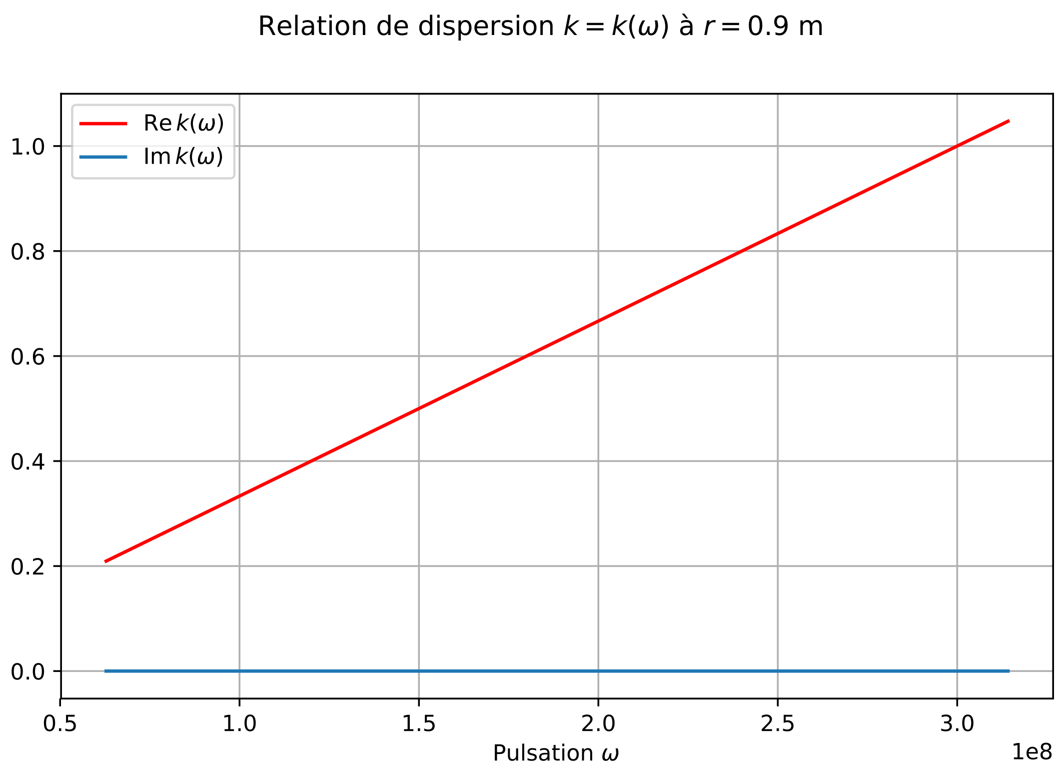
$$\begin{cases} \begin{cases} \frac{1}{c}\sqrt{\omega^2 - 1.9107554120509 \cdot 10^{16}} & \text{for}\ \omega \geq 138230076.757951 \\ -\frac{i}{c}\sqrt{-\omega^2 + 1.9107554120509 \cdot 10^{16}} & \text{otherwise} \end{cases} & \text{for}\ r > 4 \\ \frac{\omega}{c} & \text{otherwise} \end{cases}$$

La fonction suivante définit une fonction numérique correspondant à $k(\omega)$ et en fait le tracé sur l'intervalle des pulsations du courant i.

In [127]:
```python
def dispersion(puls,ra):
    """
    Fait le graphe de k = k(omega) (relation de dispersion)
    """
    wmin = np.amin(puls)
    wmax = np.amax(puls)
    wrange = np.linspace(wmin,wmax, 256)

    func = sp.lambdify((omega,c,r), k, "numpy")
    krange = func(wrange+0j, 3e8, ra)

    fig,ax = plt.subplots(1,1, figsize=(8,5))
    fig.suptitle(r"Relation de dispersion $k=k(\omega)$ à $r={}$".format(ra) \
                 + " $\mathrm{m}$")

    ax.grid(True)
    ax.plot(wrange, krange.real, 'r', label=r"$\mathrm{Re}\, k(\omega)$")
    ax.plot(wrange, krange.imag, label=r"$\mathrm{Im}\, k(\omega)$")
    ax.set_xlabel(r"Pulsation $\omega$")
    ax.legend()
    return func
```

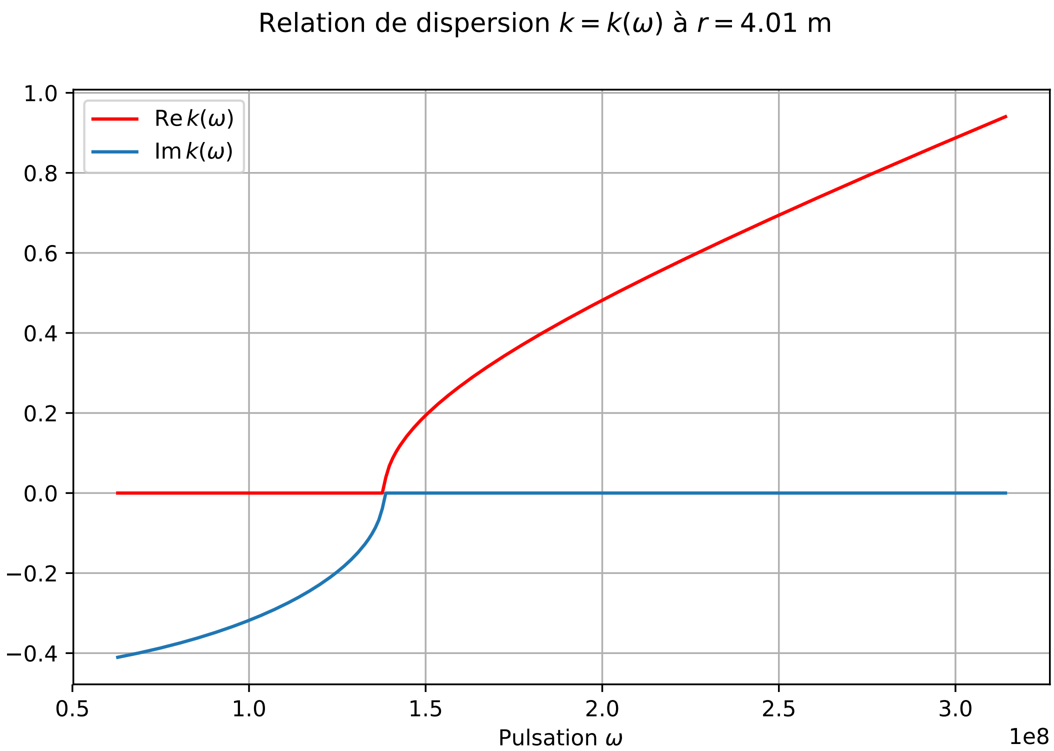
In [128]:
```python
dispersion(puls, 0.9)
```

Out[128]:
```
<function numpy.<lambda>>
```



In [129]:
```python
dispersion(puls, 4.01)
```

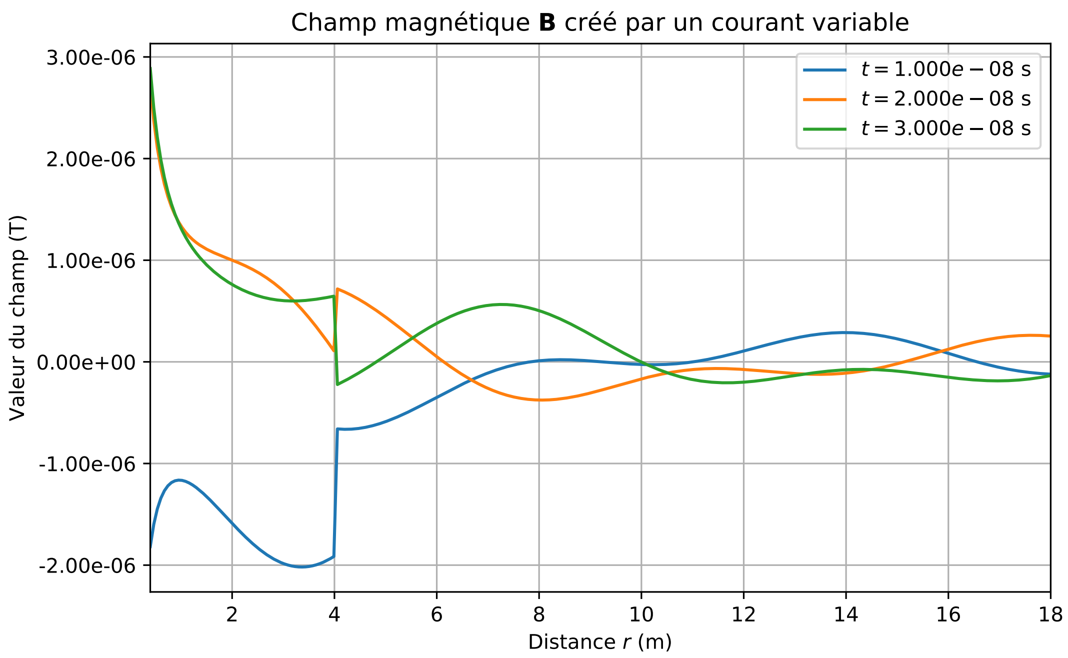
Out[129]:
```
<function numpy.<lambda>>
```



In [122]:
```
plasmaField = Field(intens, puls)
```

In [123]:
```
rmin = 0.4
rmax = 18
xmin, xmax = rmin/2**0.5, rmax/2**0.5

Omega = Domain(xmax,xmax,256,xmin,xmin)

plasmaField.profile(Omega,[1e-8,2e-8, 3e-8])
```

```
C:\Program Files\Anaconda3\lib\site-packages\numpy\core\numeric.py:482: ComplexWarning: Casting comp
  return array(a, dtype, copy=False, order=order)
```




```
In [119]: plasmaField.animate(Omega, 0,5e-7)
```

```
C:\Program Files\Anaconda3\lib\site-packages\matplotlib\transforms.py:994: ComplexWarning: Casting c
  self._points[:, 1] = interval
C:\Program Files\Anaconda3\lib\site-packages\numpy\core\numeric.py:482: ComplexWarning: Casting comp
  return array(a, dtype, copy=False, order=order)
```

```
Out[119]: <matplotlib.animation.FuncAnimation at 0x2ae30939c18>
```