```
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.

2 Implémentations en Python du champ magnétique et du courant électrique

Dans tout ce document, 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 Domaine spatial : classe `Domain`

```
In [4]: class Domain:

            def __init__(self, xm, ym, I, 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,I)
        self.I = I

        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
```

2.2 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 [5]: class Field:

            def __init__(self, intens=None, puls=None, phas=None):
                # Composante du potentiel vecteur associée à la pulsation omega
                self.A_component = -mu*Ic/4* \
                    (sp.bessely(0,k*r) + sp.I*sp.besselj(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()

                self.cel = 3e8 # Célérité des ondes ; à modifier en fonction du milieu

                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)
                    self.spectre(intens, puls)

            def spectre(self, intens, puls):
                '''
                Construit le champ magnétique
                '''
                c0 = self.cel
                mu0_v = 4e-7*np.pi

                spectral_data = zip(intens, puls)
                self.impl_modules = ['numpy',
                            {"besselj":spec.jv,
                             "bessely":spec.yv,
                             "besseli":spec.iv,
                             "sqrt":lambda x: np.sqrt(x+0j)}]

                orth_C = sum([
```

```python
            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) ])

    orth_expr = sp.re(orth_C)

    orth_func = sp.lambdify((r, t), orth_expr,
        modules = self.impl_modules)

    self.orth_C = orth_C
    self.orth_expr = orth_expr
    self.orth_func_r = orth_func

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

    self.orth_func_xy = sp.lambdify((x, y, t), orth_expr.subs({r:rxy}),
        modules = self.impl_modules)
    self.field_expr = (-y*Cart.i + x*Cart.j)*orth_expr.subs({r:rxy})/rxy
    self.field_func = sp.lambdify((x, y, t),
        self.field_expr.to_matrix(Cart),
        modules = self.impl_modules)

def legende(self,t):
    """Définit la légende"""
    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):
    '''
    Construit les graphes du champ magnétique B aux temps donnés
    dans la liste "times"
    '''
    func = self.orth_func_r
    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 animate(self, Omega, t0, t1, animtime=10, title = None):
        """
        Construit une animation du profil du champ entres les temps spécifiés.
        """

        func = self.orth_func_r
        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)

        # Paramètres d'animation
        fps = 30
        frames = int(np.ceil(fps*animtime))
        dt = (t1 - t0)/frames # Saut en temps réel entre deux images
        interval = 1000/fps # Nombre de millisecondes entre deux images

        # Cadrage
        ymax = func(radii,t0).max()
        ax.set_ylim((-1.3*ymax,1.3*ymax))

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

        def animate(i):
            ti = dt*i+t0
            champ = func(radii, ti)
            line.set_data(radii, champ)
            time_text.set_text(self.legende(ti))
            return line,

        anim = animation.FuncAnimation(fig, animate, init_func=init,
                    frames=frames, interval=interval, blit=True)
        self.anim = anim

    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
```

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

        normB = func(*Omega(), t)

        ax.plot_wireframe(*grid, normB)

        self.surf = fig

    def animate3D(self, Omega, t0, tm, fps=25):
        """
        tm -> intervalle [t0,tm]
        fps : nombre d'images par seconde à générer
        """
        animtime = 6 # durée de l'animation
        interval = 1000/fps # temps entre deux frames
        N = int(np.ceil(animtime*fps))
        dt = (tm-t0)/N # saut temporel entre chaque frame

        func = np.vectorize(self.orth_func_xy)
        grid,fig,ax = self._setup_surface(Omega)

        Bth = func(*grid, t0)
        zlims = (np.nanmin(Bth),1.5*np.nanmax(Bth))

        surf = ax.plot_wireframe(*grid, Bth)
        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
            Bth = func(*Omega(), ti)
            time_text = ax.text2D(0.5,1,self.legende(ti),
                    horizontalalignment='center',
                    transform=ax.transAxes)
            data = ax.plot_wireframe(*grid, Bth)
            return data, time_text

        anim = animation.FuncAnimation(fig,update,
                        frames=N,interval=interval)
        self.anim3D = anim
        self.animfig = fig
```

On pourrait éventuellement implémenter une représentation du champ électrique… Le lecteur intrépride pourra s'y aventurer en introduisant une sous-classe de `Field` ou en la modifiant.

In [6]:
```python
# Composante du champ électrique associée à la pulsation omega
A_component = Field().A_component
-sp.diff(A_component, t)
```

Out[6]:

$$\frac{iI}{4}\mu_0\omega\left(iJ_0\left(kr\right)+Y_0\left(kr\right)\right)e^{i\omega t}$$

2.3 Courant électrique : classe current

La cellule suivante définit les courants électriques comme une classe Python `Current`, dont les attributs sont notamment l'expression formelle du champ (`expr`), et la fonction numérique qui permet de calculer le courant à un instant (`func`).

```
In [8]: class Current:
            # Composante du courant électrique de pulsation omega
            cour_component = Ic*sp.exp(sp.I*omega*t)
            cour_component
            def _spectre(self, intens):
                """
                Calcule l'expression mathématique 'self.expr' et définit une fonction
                numérique 'self.func' permettant de calculer le courant à un instant.
                """
                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 _expr(self, expr):
                """
                Définit le courant selon son expression."""
                self.expr = expr
                self.func = sp.lambdify(t, expr, modules=['numpy'])

            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 not(intens is None):
                    if not(phas is None):
                        intens = np.asarray(intens)*np.exp(1j*np.asarray(phas))

                    self.intensities = intens
                    self.pulsations = puls
                    self.frequences = self.pulsations/(2*np.pi)

                    self._spectre(intens)

            def fft(self, fs, N):
                dt = 1/fs
                sample_time = np.linspace(-N*dt,N*dt,N+1)
                samples = self.func(sample_time)
```

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

    def draw(self, tmin, tmax, N=1000, custTitle=None):
        """
        Construit la représentation graphique de la fonction i(t),
        stockée dans l'attribut 'self.graphe'

        custTitle : titre optionnel à fournir
        """
        times = np.linspace(tmin, tmax, N)

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

        if custTitle:
            ax.set_title(custTitle)
        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 draw_fft(self):
        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.abs(self.intensities))

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

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

        self.graphe_fft = fig
```

3 Exemples d'utilisation

3.1 Données initiales

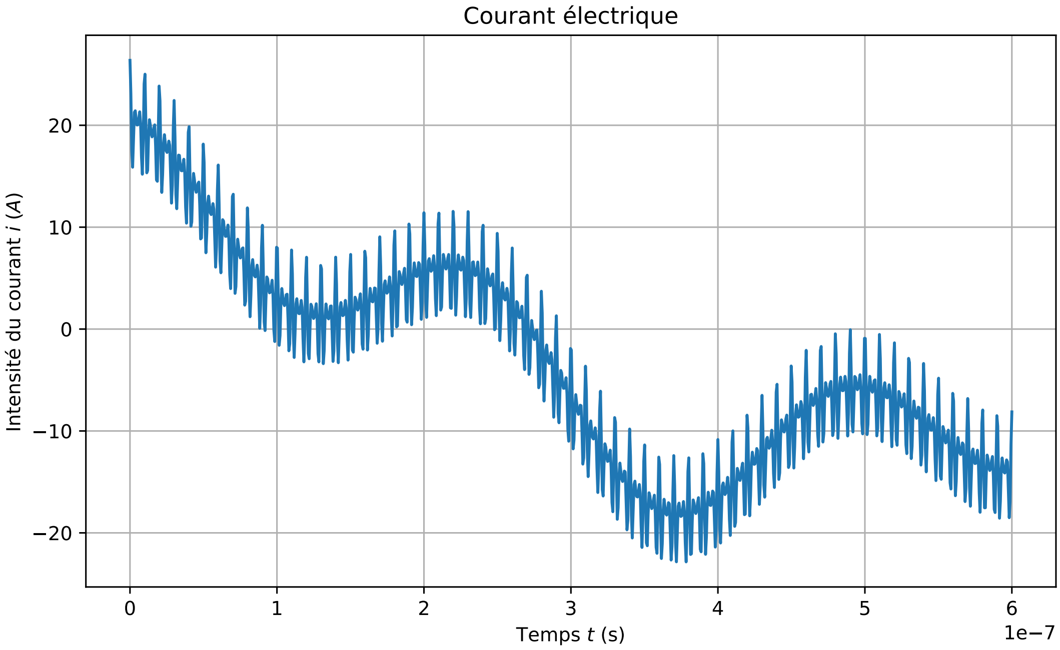
Entrez dans la variable `freqs` les fréquences du courant voulu, et dans `phas` les phases. Exécutez la cellule (`Ctrl + Entrée` sur le clavier) pour définir la fonction de champ :

On va tester avec une autre relation de dispersion:

```python
In [9]: k = omega/c # relation de dispersion

        freqs = np.array([n*1e8 for n in range(2,5)] + \
            [n*1e6 for n in [1,1.1,4,8]])
```

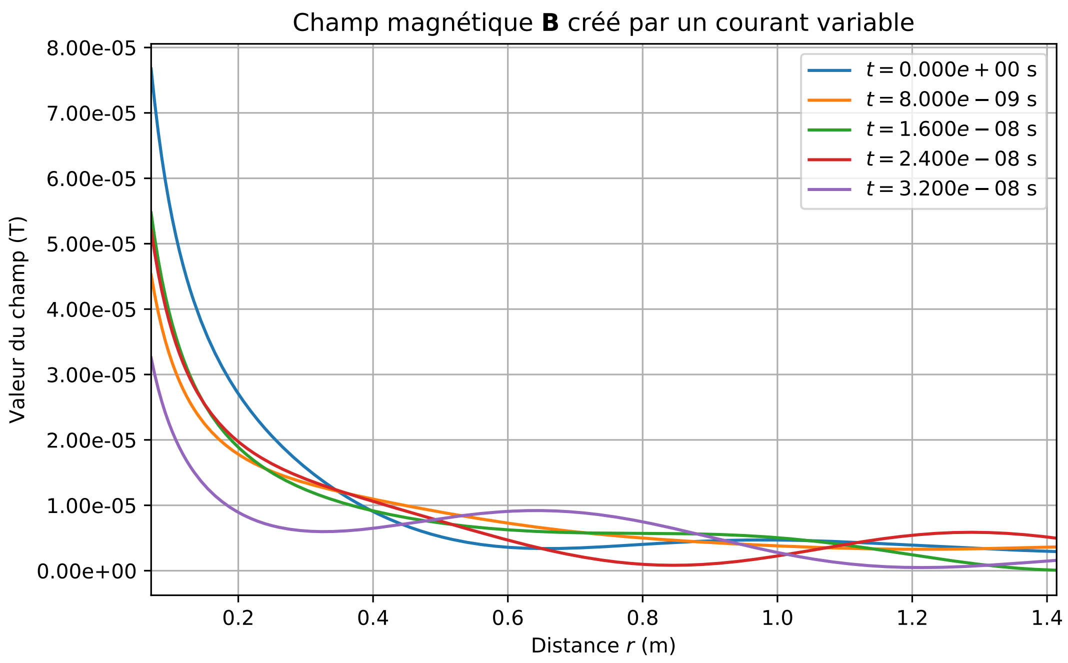
```
puls = 2*np.pi*freqs # Pulsations associées
intens = np.array([2,3,1,7,7,7]) # Intensités des composantes

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

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

```
In [10]: courant = Current(intens, puls, phases)
         courant.draw(0,6e-7)
```

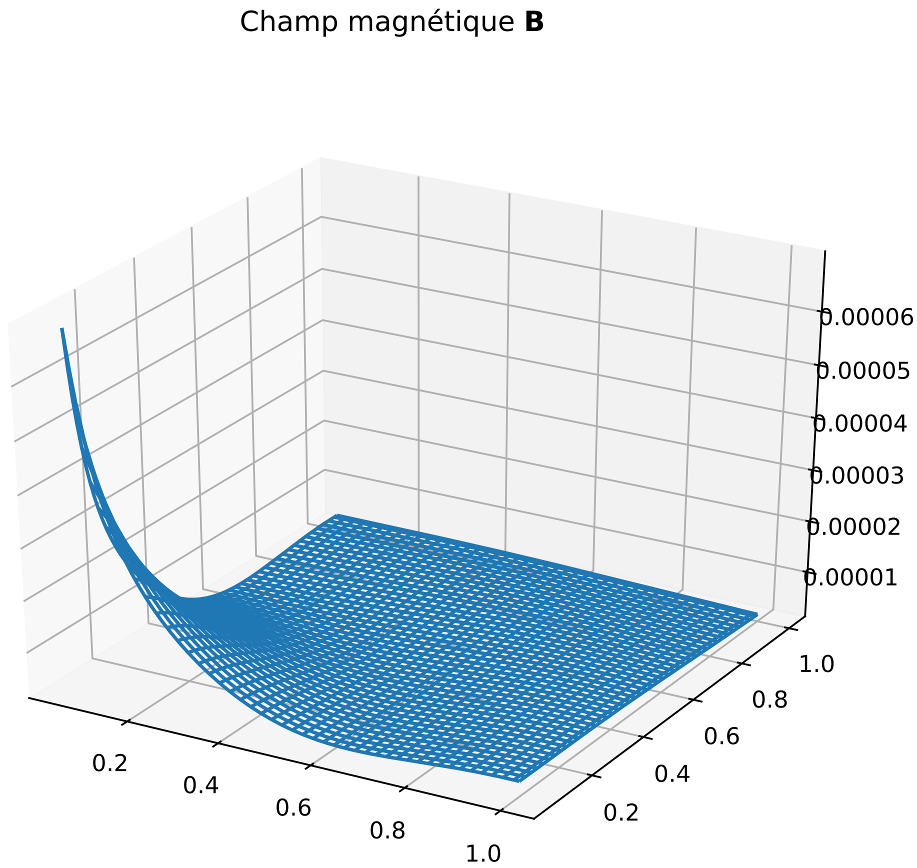


La cellule suivante définit les distances minimale et maximale pour lesquels tracer le profil du champ magnétique :

```
In [11]: Omega = Domain(1,1,256,256, 0.05, 0.05)

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

         B_field.profile(Omega,times)
```




```
In [12]: B_field.surface(Omega, 1e-6)
```



3.2 Animations

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

```
In [ ]: Omega = Domain(1,1,200,200, 0,0,eps=0.03)
        times_an = (0, 5e-6) # (t0, tm)

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

Et visualiser cela en trois dimensions:

```
In [ ]: Omega = Domain(2,2,64,64,0,0,eps=0.03)
        B_field.animate3D(Omega, *times_an)

In [ ]: B_field.anim3D.save('onde.mp4')
```

4 Exemple d'application: Paquet d'ondes

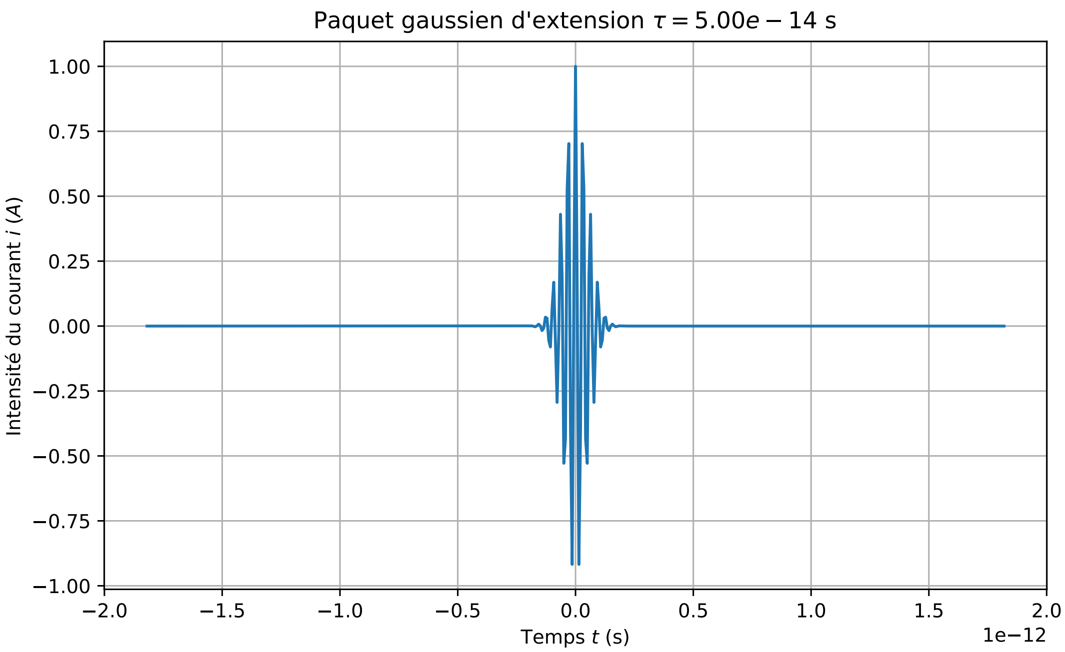
```
In [13]: def gaussienne(tau):
             tard = t - 0
             expr = sp.exp(-tard**2/(2*tau**2))*sp.cos(10*t/tau)
             out = Current()
             out._expr(expr)
             return out

In [14]: tau = 5e-14
         N = 2**9 # Nombre d'échantillons
         fs = 2**48 # Fréquence d'échantillonnage

         courant = gaussienne(tau)
         courant.expr
```

Out[14]:

$$\frac{\cos\left(200000000000000.0t\right)}{e^{2.0\cdot 10^{26}t^2}}$$

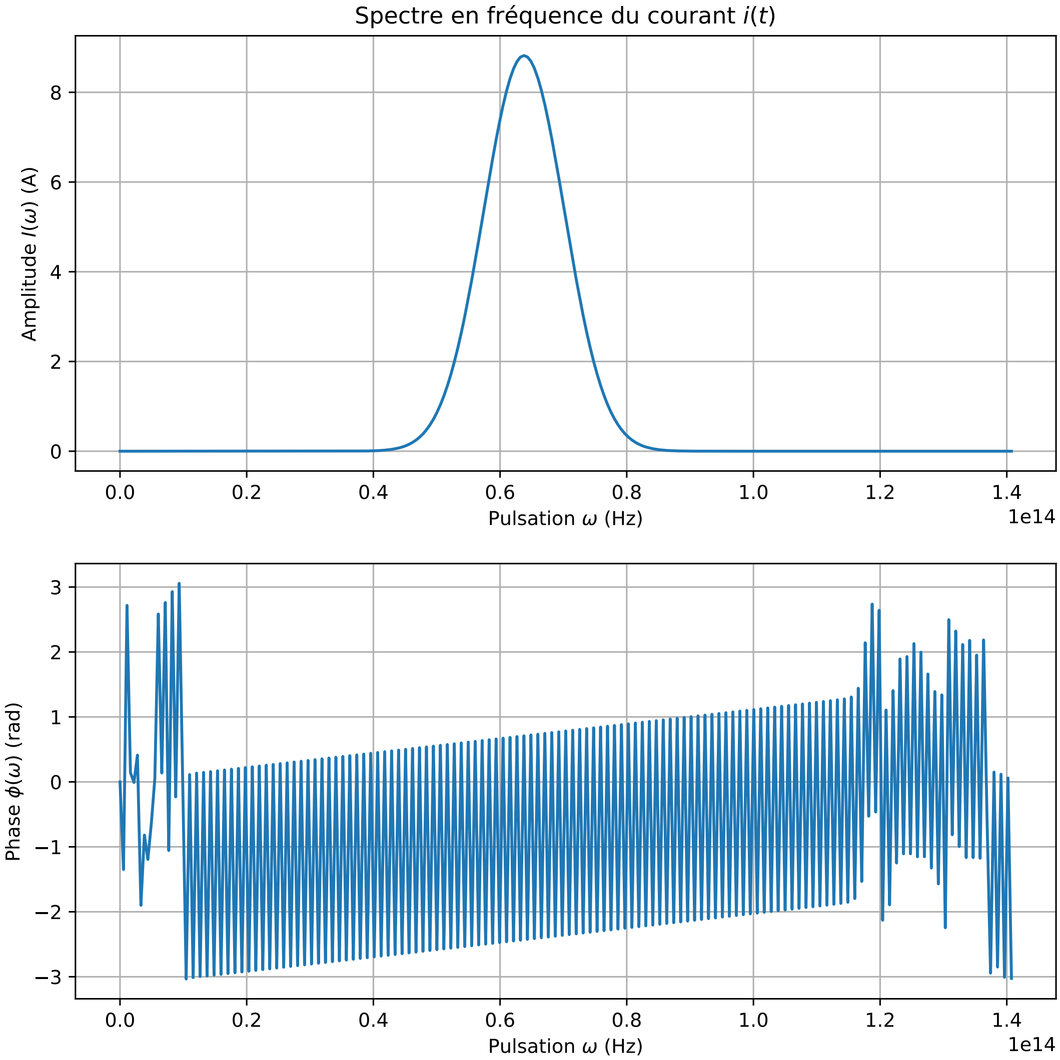
Courbe représentative du courant $i(t) = e^{-t^2/(2\tau^2)}\cos\left(\frac{t}{\tau}\right)$:

```
In [15]: titros = r"Paquet gaussien d'extension $\tau = {:.2e}$ s".format(tau)
         courant.draw(-N/fs,N/fs, N+1, titros)
```



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

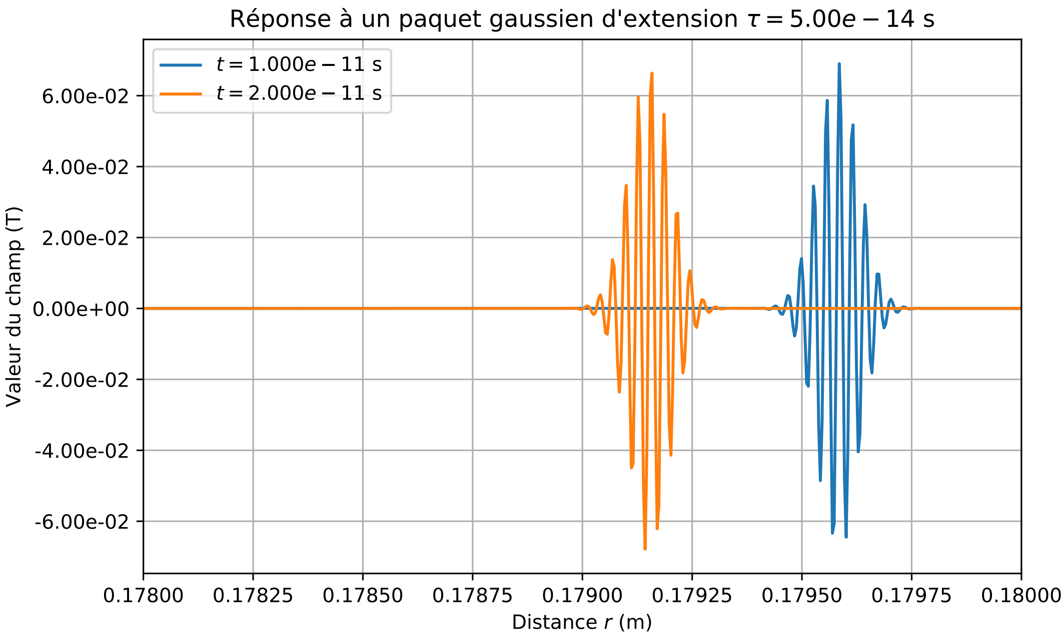
```
In [16]: courant.fft(fs, N)
         courant.draw_fft()
```




```
In [17]: intens = courant.intensities
         puls = courant.pulsations

         k = omega/c

         B_field = Field(intens, puls)
```

```
In [18]: rmin = 0.178
         rmax = 0.18
         xmin = rmin/2**0.5
         xmax = rmax/2**0.5

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

```
times = [1e-11*k for k in [1,2]]
titre_gauss = "Réponse à un paquet gaussien d'extension " + \
        r"$\tau={:.2e}$".format(tau) + " $\mathrm{s}$"
B_field.profile(Omega, times, title=titre_gauss)
```




```
In [ ]: times = (0,2e-11)
        Omega = Domain(xmax,xmax,512,512,xmin,xmin,False)

        B_field.animate(Omega,*times,
            title="Propagation d'un paquet gaussien d'extension " + \
                    r"$\tau = {:.2e}$".format(tau) + \
                    "$\mathrm{s}$")

        B_field.anim
```

```
In [ ]: times = (0,2e-11)
        Omega = Domain(xmax,xmax,64,64,xmin,xmin,False)
        B_field.animate3D(Omega, *times)

        B_field.anim3D.save('surface_paquet.mp4')
```

5 En milieu dispersif

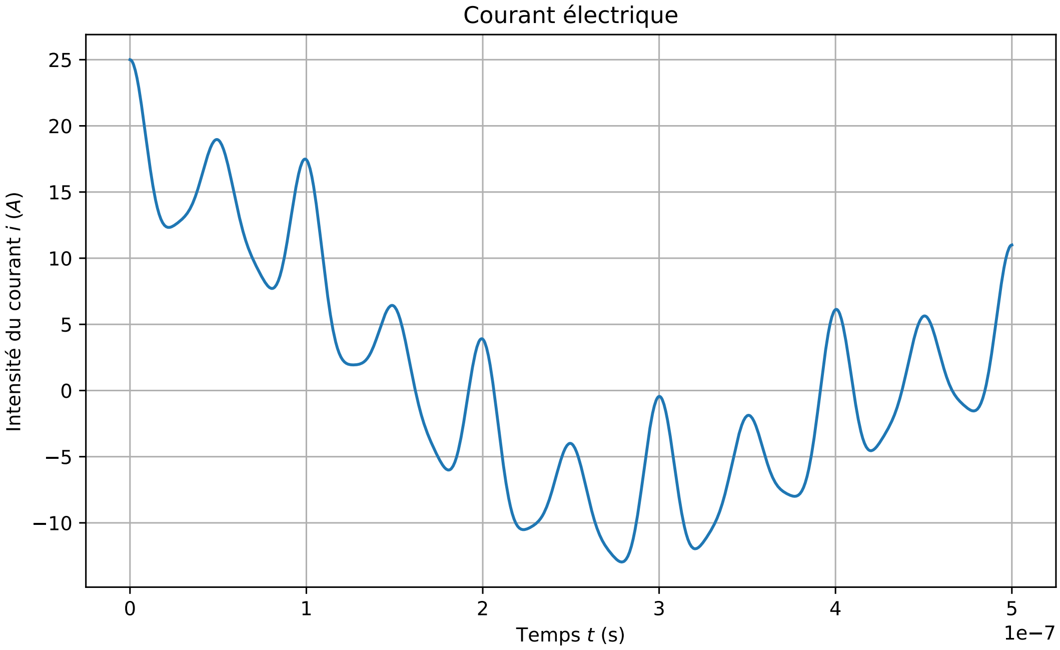
On s'intéresse à la propagation dans un plasma. La relation de dispersion (entre vecteur d'onde k et pulsation ω) dans un plasma de fréquence ω_0 est

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

```
In [37]: freqs = np.array([n*1e7 for n in range(5)] + \
                          [n*1e6 for n in [1,2]])
         puls = 2*np.pi*freqs

         puls
```

```
intens = np.array([1,1,5,1,1,7,9])

courant = Current(intens, puls)
```

In [39]:
```
courant.draw(0,5e-7)
```



In [20]:
```
omega0 = 5.6e7

kplas = sp.Piecewise((sp.sqrt(omega**2-omega0**2)/c, omega>=omega0),
                     (-sp.I*sp.sqrt(omega0**2-omega**2)/c, True ))
kplas
```

Out[20]:

$$\begin{cases} \frac{1}{c}\sqrt{\omega^2 - 3.136 \cdot 10^{15}} & \text{for } \omega \geq 56000000.0 \\ -\frac{i}{c}\sqrt{-\omega^2 + 3.136 \cdot 10^{15}} & \text{otherwise} \end{cases}$$

In [21]:
```
k = sp.Piecewise((omega/c, r > 1), (kplas, True))
k
```

Out[21]:

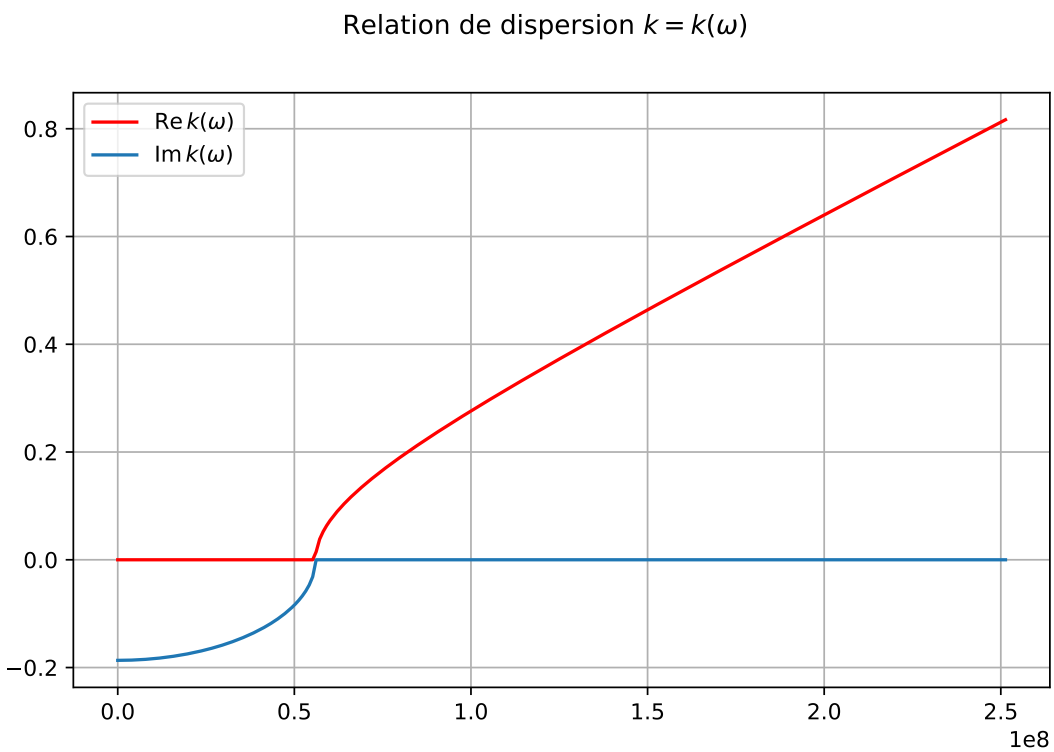
$$\begin{cases} \frac{\omega}{c} & \text{for } r > 1 \\ \begin{cases} \frac{1}{c}\sqrt{\omega^2 - 3.136 \cdot 10^{15}} & \text{for } \omega \geq 56000000.0 \\ -\frac{i}{c}\sqrt{-\omega^2 + 3.136 \cdot 10^{15}} & \text{otherwise} \end{cases} & \text{otherwise} \end{cases}$$

In [44]:
```
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)
```

```python
    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 $k=k(\omega)$")

    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.legend()
    return func
```

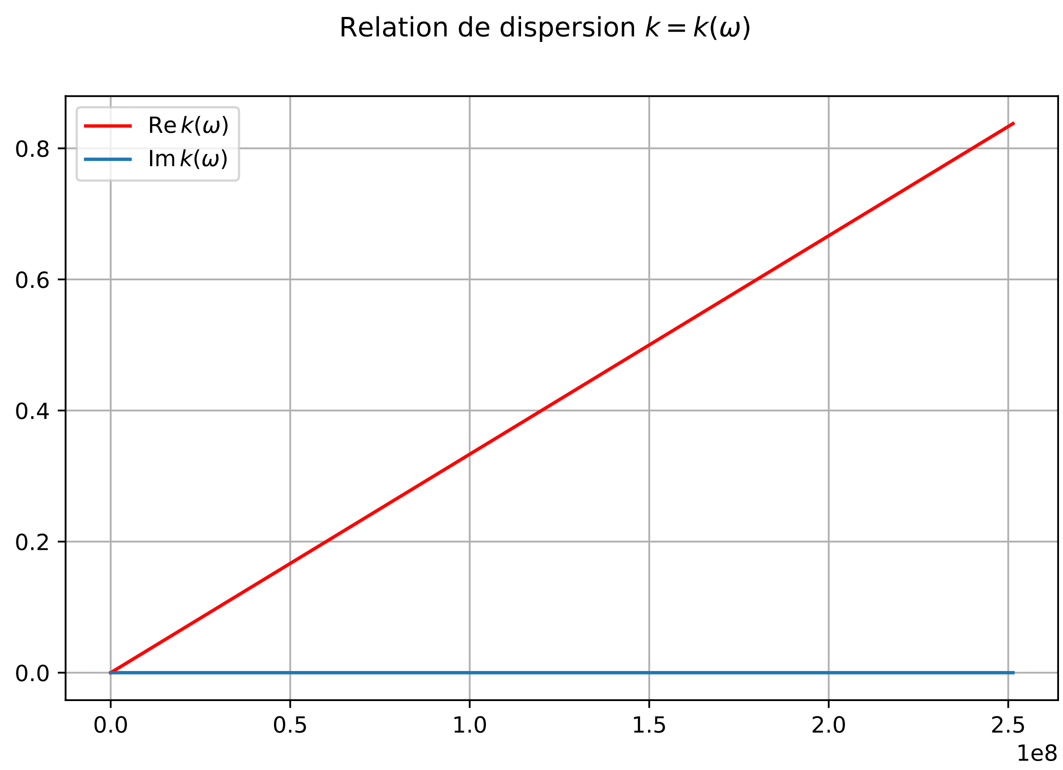
In [45]: `dispersion(puls, 1.0)`

Out[45]: `<function numpy.<lambda>>`

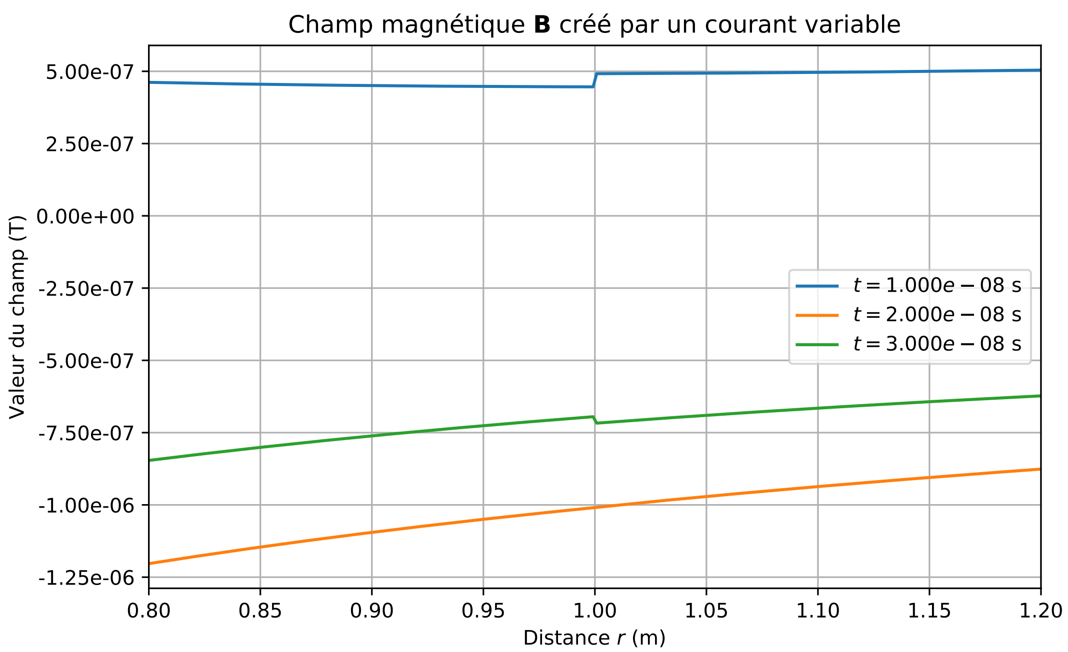


In [46]: `dispersion(puls, 1.01)`

Out[46]: `<function numpy.<lambda>>`




```
In [32]: B_field = Field(intens, puls)

In [52]: rmin = 0.8
         rmax = 1.2
         xmin, xmax = rmin/2**0.5, rmax/2**0.5

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

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



6 Théorie

Le champ magnétique $\mathbf{B}$ dérive d'un champ $\mathbf{A}$ appelé potentiel vecteur : $\mathbf{B} = \nabla \wedge \mathbf{A}$. Par symétrie cylindrique, on a $\mathbf{B}(\mathbf{r}, t) = B(r, t)\mathbf{e}_\theta$, puis $\mathbf{A}(\mathbf{r}, t) = A(r, t)\mathbf{e}_z$.

Le potentiel vecteur $\mathbf{A} = A(r, t)\mathbf{e}_z$ est solution de l'équation d'onde

$$\Delta \mathbf{A} - \frac{1}{c^2}\frac{\partial^2 \mathbf{A}}{\partial t^2} = -\mu_0 \mathbf{J}(r, t), \tag{1}$$

avec $\mathbf{J}(\mathbf{r}, t) = i(t)\delta(r)\delta(\theta)\mathbf{e}_z$ la densité volumique de courant, de sorte que pour toute surface (Σ) traversée par le fil, on ait que le flux de $\mathbf{J}$ soit égal au courant parcourant le fil : $\iint_{(\Sigma)} \mathbf{J}(\mathbf{r}, t) \cdot \mathrm{d}\sigma = i(t)$.

Pour un courant sinusoïdal $i(t) = I\exp(i\omega t)$, le potentiel s'écrit $\mathbf{A}(\mathbf{r}, t) = f(r)\exp(i\omega t)\mathbf{e}_z$ et l'équation aux dérivées partielles se réduit à

$$\frac{1}{r}\frac{\mathrm{d}}{\mathrm{d}r}\left(r\frac{\mathrm{d}f}{\mathrm{d}r}\right) + k^2 f(r) = -\frac{\mu_0 I}{2\pi r}\delta(r) \tag{2}$$

avec $k = \dfrac{\omega}{c}$.

La solution prend la forme

$$f(r) = -\frac{\mu_0 I}{4}\left(Y_0(kr) + iJ_0(kr)\right)$$

où J_0, Y_0 sont les 0-ièmes fonctions de Bessel de la première et seconde espèce, solutions de

$$xy''(x) + y'(x) + xy(x) = 0.$$