```
import numpy as np
import numpy.fft as fft
import matplotlib.pyplot as plt
```

- Durée de l'épreuve 30 minutes.
- Les questions sont notées indépendamment les unes des autres.
- Tous les documents ainsi que les calculatrices et les **téléphones portables** sont interdits. Ils doivent être éteints.
- La communication entre les étudiants est interdite.
- Toutes vos fonctions doivent être commentées selon la norme de `python`.

- Le sujet comporte 4 questions, dont la première est théorique. Le barème est donné à titre indicatif est peut être modifié.

Résolution de l'équation de Schrödinger (linéaire) à l'aide de série de Fourier et discrétisation par FFT

Résultats théoriques

Nous nous intéressons à la résolution de l'équation de Schrödinger sans terme source avec conditions de bord périodiques : \begin{equation} \begin{cases} & \frac{\partial u}{\partial t} - i \frac{\partial^2 u}{\partial x^2} = 0 \quad \forall (t,x) \in]0;+\infty[\times [0 ; 2 \pi] \quad , \\ & u(t=0,.) = u_0 \quad, \\ & u(t,0) = u(t,2\pi) \quad , \end{cases} \end{equation} avec u_0 de classe C^4.

On admet le calcul suivant obtenu par calcul à l'aide des séries de Fourier : Si u est une solution régulière de l'équation précédente, alors,

$$u(t,x) = \sum_{k=-\infty}^{+\infty} c_k(u_0) e^{-ik^2 t} e^{ikx}$$

Question 1 (5 points ; à faire sur la copie)

Montrer que si u_0 est de classe C^4, alors :

- la série $\sum_{k=-\infty}^{+\infty} c_k(u_0) e^{-ik^2 t} e^{ikx}$ est bien définie et est au moins de classe C^2,
- elle est bien solution de :

$$\frac{\partial u}{\partial t} - i\frac{\partial^2 u}{\partial x^2} = 0.$$

On admettra qu'elle vérifie les conditions limites périodiques et qu'elle converge vers la donnée initiale quand le temps tend vers 0.

=== BEGIN MARK SCHEME ===

u_k = c_k[u_0] e^{-ik^2t} e^{ikx}

1pt - u0 C^4, donc |c_k[u_0] | <= C/k^4

1pt - la série somme(u_k) est NCV car |u_k| < C/k^4

2pt - idem pour les dérivées première + secondes (majoration en cst/k^{4-nb dérivées}),

1pt - sol car permutation somme/dérivation et u_k est sol

=== END MARK SCHEME ===

=== BEGIN MARK SCHEME ===

Correction Question 1

Étape 1 : Majoration des coefficients de Fourier

Puisque u_0 est de classe C^4 et 2π-périodique, on intègre par parties 4 fois dans la définition des coefficients de Fourier (les termes de bord s'annulent par périodicité) :

$$c_k[u_0] = \frac{1}{2\pi}\int_0^{2\pi} u_0(x)e^{-ikx}dx = \frac{1}{(ik)^4}c_k\left[u_0^{(4)}\right] \quad (k \neq 0)$$

Or $u_0^{(4)}$ est continue sur $[0, 2\pi)$ donc ses coefficients de Fourier sont bornés : $\exists M > 0, \left|c_k\left[u_0^{(4)}\right]\right| \leq M$ pour tout k. On obtient :

$$\left|c_k[u_0]\right| \leq \frac{C}{k^4}$$

Étape 2 : Convergence normale de la série et de ses dérivées

Posons $u_k(t,x) = c_k[u_0]e^{-ik^2t}e^{ikx}$. Comme $\left|e^{-ik^2t}\right| = 1$ (exposant purement imaginaire) :

$$\left|u_k(t,x)\right| = \left|c_k[u_0]\right| \leq \frac{C}{k^4}$$

La série $\sum 1/k^4$ converge (Riemann, $p = 4 > 1$), donc $\sum_k u_k$ **converge normalement** $\Rightarrow$ la somme u est bien définie et continue.

Pour les dérivées, on majore terme à terme :

Dérivée	Majorant	Série convergente ?
$\partial_t u_k = -i k^2 u_k$	$k^2 \cdot C/k^4 = C/k^2$	$(\sum 1/k^2 < \infty)$
$\partial_x u_k = i k u_k$	$k \cdot C/k^4 = C/k^3$	$(\sum 1/k^3 < \infty)$
$\partial_x^2 u_k = -k^2 u_k$	$k^2 \cdot C/k^4 = C/k^2$	$(\sum 1/k^2 < \infty)$

Les séries des dérivées convergent normalement ⇒ on peut **dériver terme à terme** ⇒ u est au moins de classe C^2.

Étape 3 : u est solution de l'EDP

Chaque terme u_k est solution : on vérifie directement

$$\partial_t u_k - i\partial_x^2 u_k = \left(-i k^2\right) u_k - i\left(-k^2\right) u_k = -i k^2 u_k + i k^2 u_k = 0$$

Puisque les séries des dérivées convergent normalement, on peut **permuter somme et dérivation** :

$$\partial_t u - i\partial_x^2 u = \sum_{k=-\infty}^{+\infty} \left(\partial_t u_k - i\partial_x^2 u_k\right) = \sum_{k=-\infty}^{+\infty} 0 = 0 \blacksquare$$

=== END MARK SCHEME ===

Discrétisation

On admet que le résultat théorique obtenu à partir des séries de Fourier peut s'adapter à la transformée discrète :

Remarque : Les coefficients $p_k[u_0)$ sont calculés par la FFT et les fonctions numpy associées. $\frac{p_k[u_0)}{2m}$ *est une approximation du coefficients de Fourier $c_k[u_0)$ (qui converge très vite).*

Question 2 (4+1 points)

- Proposez une fonction `schrodinger_fft` qui :
  - prend en argument :
    - un tableau `u0` contenant la donnée initiale : `u0[i]` est la valeur de la donnée initiale au point `x[i]`,
    - un pas de temps `dt` (float),
    - un temps final `T` (float),
  - construit `time` une subdivision régulière en temps de $[0\,;T f)$ à $N t$ points,
  - crée un tableau `u` de dimension 2 à `Nt` lignes et `Nx` colonnes **à valeurs complexes**,
  - calcule la TFD (en espace) de la donnée initiale dans une variable `spectre` (attention spectre est à valeur complexes),

  - résout l'équation des ondes dans l'espace spectral à chaque pas de temps t_n (à l'aide de la formule ci-dessus sur P_m),
  - renvoie le couple `time, u`
- Testez votre fonction pour $u_0 : x \mapsto \sin(x)$ et vérifier que vous obtenez des résultats similaire au graphique ci-dessous.

Indications :

- *pour créer un tableau complexe, il faut ajouter l'argument optionnel* `dtype=complex` *(par exemple* `A = np.zeros(10,dtype=complex)`*),*
- *pour accéder à la partie réelle d'un nombre (ou d'un tableau), il faut faire* `C.real`*; de même la partie imaginaire s'obtient via* `C.imag`*,*
- *le nombre* i *tel que* $i^2 = -1$ *se tappe* `1.j` *en python.*

solution numerique de l'equation des ondes

=== BEGIN MARK SCHEME ===

Initialisation (time, u, spectre) => 1,5 points

boucle en temps + calcul u[n+1] => 2 points

resultat renvoyé et autres : 0,5 points

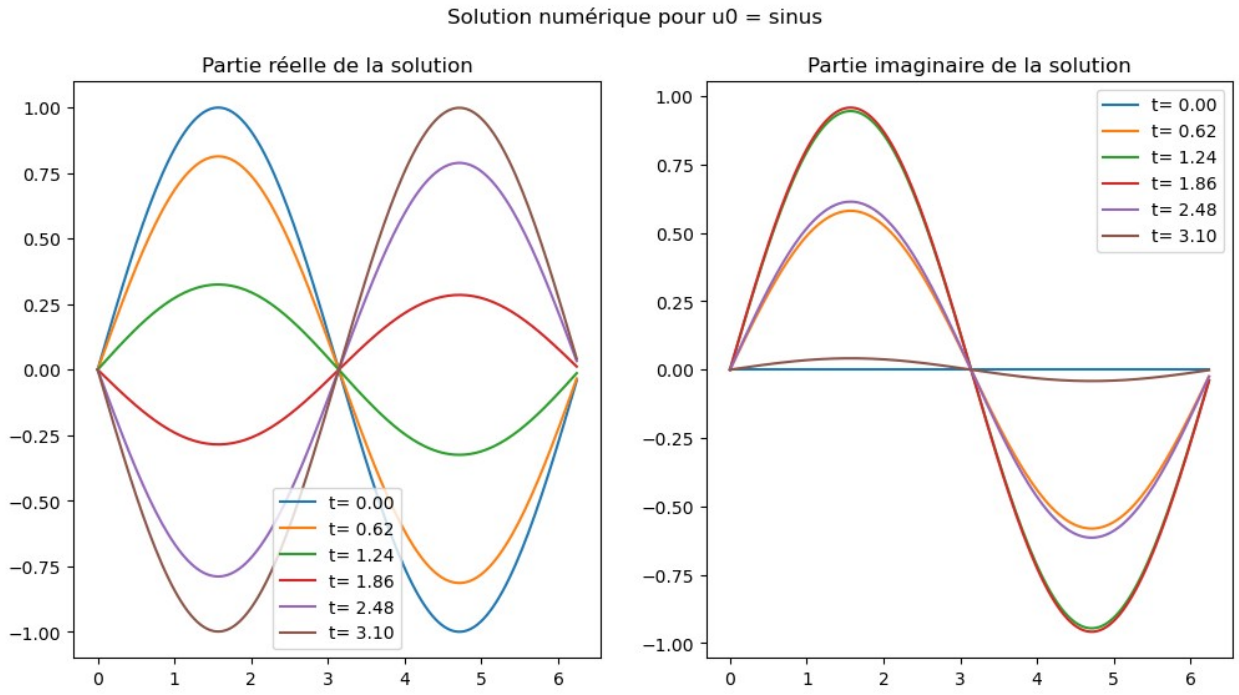
test : 1 points (dont légendes et axes)

=== END MARK SCHEME ===

```
def schrodinger_fft(u0,dt,T):
    """Résout l'équation de Schrödinger (d/dt)[u] - i*(d^2/dx^2)[u] =
0
    pour t dans [0;T] et x dans [0;2pi]

    Parameter :
    -----------
    u0 : tableau tel que u0[i] = donnée initiale en x[i]
    dt : float, pas de temps
    T  : float, temps final pour la résolution de l'équation
    Return :
    --------
    temps : tableau tel que temps[n] soit le temps tn = n*dt
    u  : tableau de dimension (Nt,Nx) (Nx = taille de u0)
            tel que u[n,i] soit une approximation de la solution en
(temps[n],x[i])
    """
    Nt = int(T/dt)+1
    Nx = u0.size
    ### BEGIN SOLUTION
    time = np.empty(Nt); time[0]=0
    u = np.empty([Nt,Nx],dtype=complex)
    spectre = fft.fft(u0)
```

```
    u[0]=u0
    coeff = np.exp(-1.j*np.square(fft.fftfreq(Nx,1/Nx))*dt)
    for n in range(Nt-1) :
        spectre *=coeff
        time[n+1]=time[n]+dt
        u[n+1] = fft.ifft(spectre)
    ### END SOLUTION
    return time, u


# Mettre votre test ici
### BEGIN SOLUTION
x = np.linspace(0,2*np.pi,150,endpoint=False)
u0=np.sin(x)
dt = 0.01
T = np.pi
step = int(T/(5*dt))

time, u = schrodinger_fft(u0,dt,T)
print(time.shape,u.shape)
fig, ax =plt.subplots(1,2,figsize=(12,6))
for i in range(0,int(T/dt),step) :
    ax[0].plot(x,u[i,:].real,label=f"t= {time[i]:.2f}")
    ax[1].plot(x,u[i,:].imag,label=f"t= {time[i]:.2f}")
ax[0].set_title("Partie réelle de la solution")
ax[0].legend()
ax[1].set_title("Partie imaginaire de la solution")
ax[1].legend()
fig.suptitle("Solution numérique pour u0 = sinus")
plt.savefig("resultat_sinx.png")
### END SOLUTION
```

```
(315,) (315, 150)
```



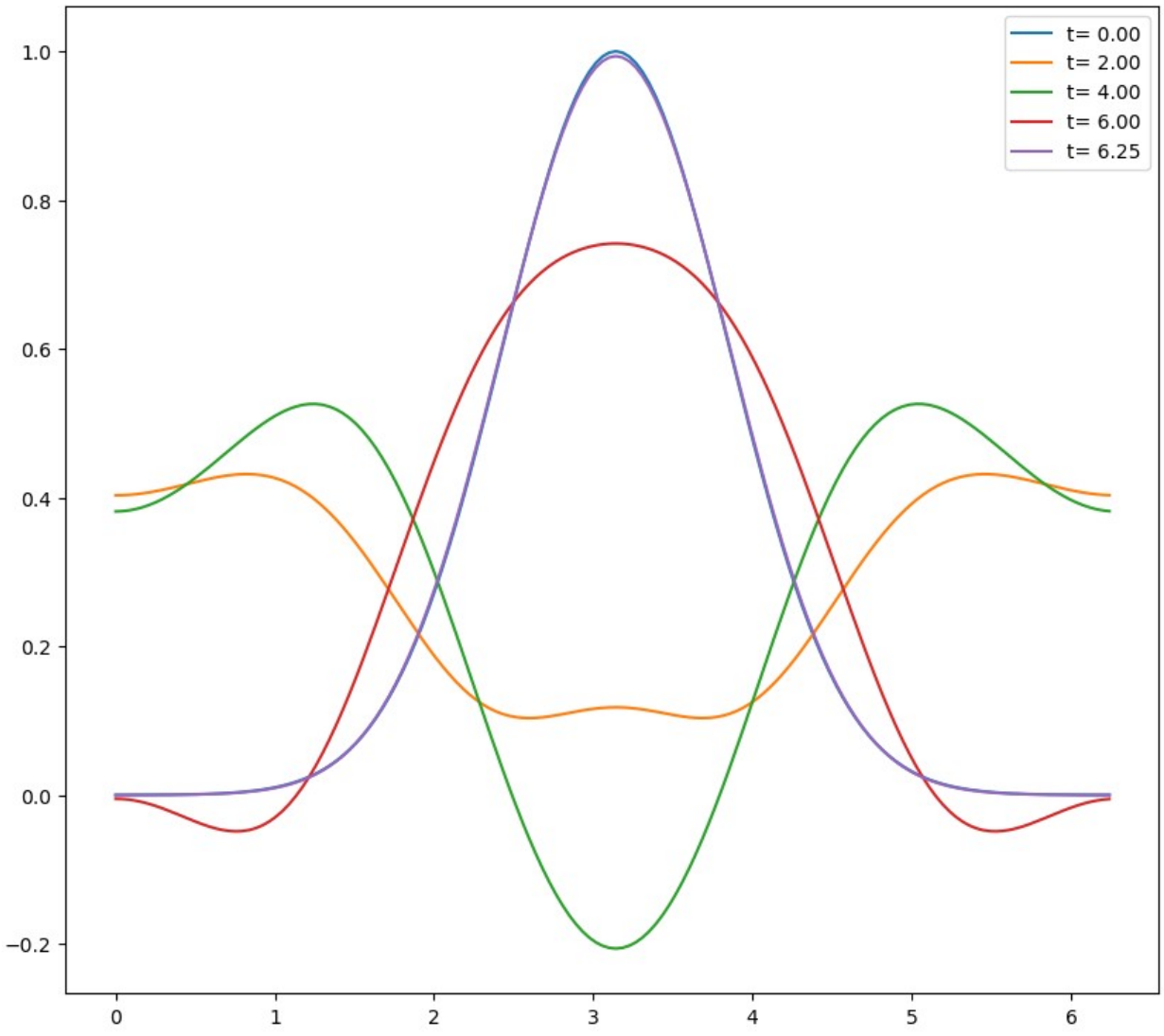
Question 3 (1 points)

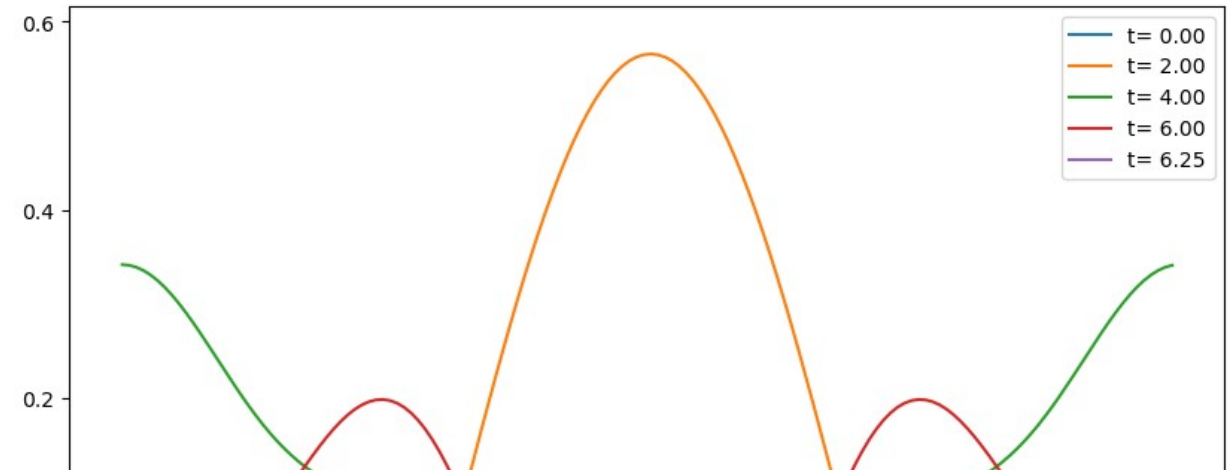
- Coder une fonction `gauss` qui prend en argument x et retourne $e^{-(x-\pi)^2}$.
- Tester votre fonction `schrodinger_fft` sur cette nouvelle donnée initiale.

```
def gauss(x):
    ### BEGIN SOLUTION
    c = np.pi
    return np.exp(-(x-c)**2)
    ### END SOLUTION

# Tests de validation
assert(np.allclose(gauss(np.pi),1))
x = np.linspace(0,2*np.pi,6)
res = np.array([5.17231862e-05, 2.86369458e-02, 6.73825451e-01,
6.73825451e-01, 2.86369458e-02, 5.17231862e-05])
assert(np.all(np.allclose(gauss(x),res)))

# Mettre votre résolution de l'équation des ondes ici et le graphique
de la solution.
x = np.linspace(0,2*np.pi,150,endpoint=False)
### BEGIN SOLUTION
u0=gauss(x)
dt = 0.05
T = 2*np.pi
step = 40

time, u = schrodinger_fft(u0,dt,T)
fig, ax =plt.subplots(2,figsize=(10,20))
```

```
for i in range(0,int(T/dt),step) :
    ax[0].plot(x,u[i,:].real,label=f"t= {time[i]:.2f}")
    ax[1].plot(x,u[i,:].imag,label=f"t= {time[i]:.2f}")
ax[0].plot(x,u[-1,:].real,label=f"t= {time[-1]:.2f}")
ax[0].legend()
ax[1].plot(x,u[-1,:].imag,label=f"t= {time[-1]:.2f}")
ax[1].legend()

### END SOLUTION
```

```
<matplotlib.legend.Legend at 0x1796a87d0>
```


t= 0.00
t= 2.00
t= 4.00
t= 6.00
t= 6.25
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
1
2
3
4
5
6


t= 0.00
t= 2.00
t= 4.00
t= 6.00
t= 6.25
0.6
0.4
0.2