Le module `matplotlib` pour faire des graphiques

Dans cette séance, nous allons apprendre à manipuler `numpy` et `matplotlib` pour tracer des graphiques.

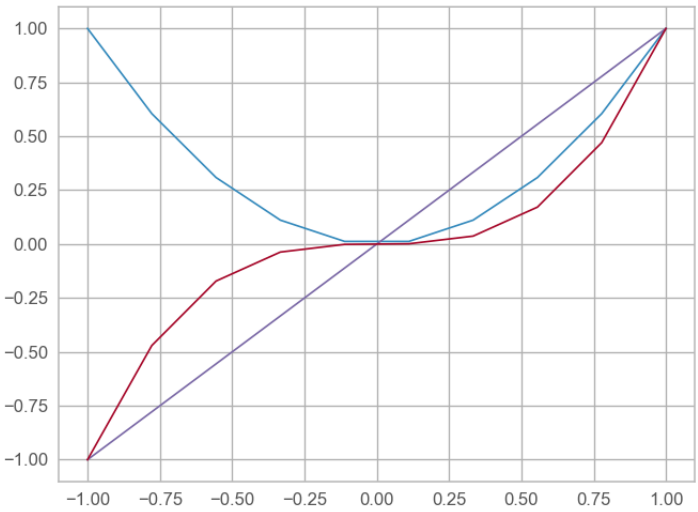
In [21]:
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

%matplotlib notebook
mpl.rcParams['savefig.dpi'] = 80
mpl.rcParams['figure.dpi'] = 60
```

Un premier exemple

`matplotlib` est un module qui permet de tracer facilement des nuages de points. Pour tracer une fonction, nous nous contenterons de relier ces points par des traits et de prendre suffisamment de points pour que l'oeil voit une courbe continue.

Pour tracer la courbe $x \mapsto x^2$ entre 0 et 1, nous procédons ainsi :

In [24]:
```
x = np.linspace(-1, 1, 10)       # abscisses des points
y = x*x                          # ordonnées des points

fig = plt.figure()               # création d'un objet figure
ax = fig.add_subplot(1, 1, 1)    # création d'un Axes (calque) dans
la figure
ax.grid(True)
ax.plot(x, y)                    # on trace le nuage de point de coo
rdonnées (x, y)
ax.plot(x, x)
ax.plot(x, x**3)
```



Out[24]: [<matplotlib.lines.Line2D at 0x7f9fa0fbc430>]

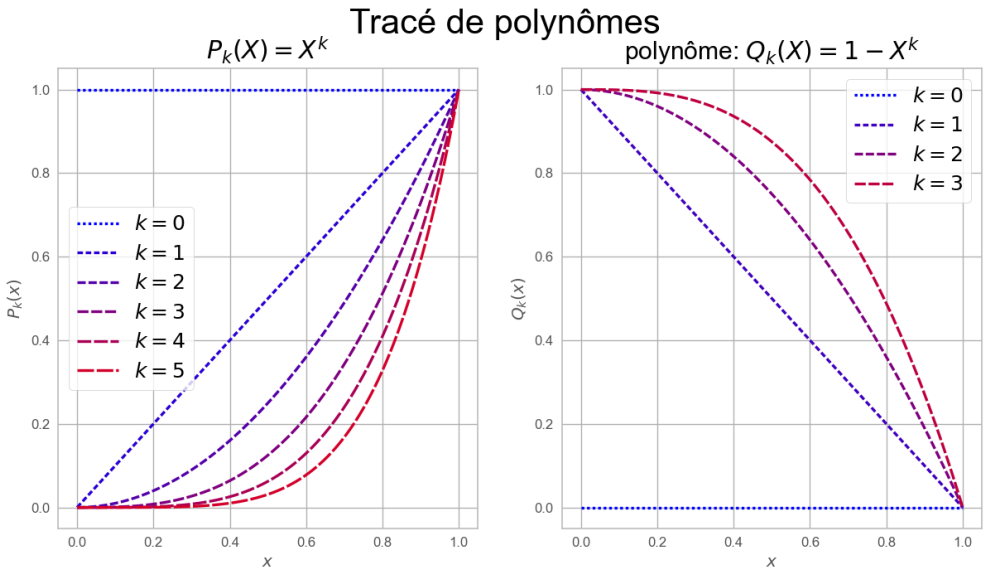
Une figure complète avec légende et titre

On introduit à présent des commandes pour ajouter un titre, une légende...

In [6]:
```
for k in range(10):
    print(f"| k pi = {np.pi*k:07.4f} | k = {k}")
```

```
| k pi = 00.0000 | k = 0
| k pi = 03.1416 | k = 1
| k pi = 06.2832 | k = 2
| k pi = 09.4248 | k = 3
| k pi = 12.5664 | k = 4
| k pi = 15.7080 | k = 5
| k pi = 18.8496 | k = 6
| k pi = 21.9911 | k = 7
| k pi = 25.1327 | k = 8
| k pi = 28.2743 | k = 9
```

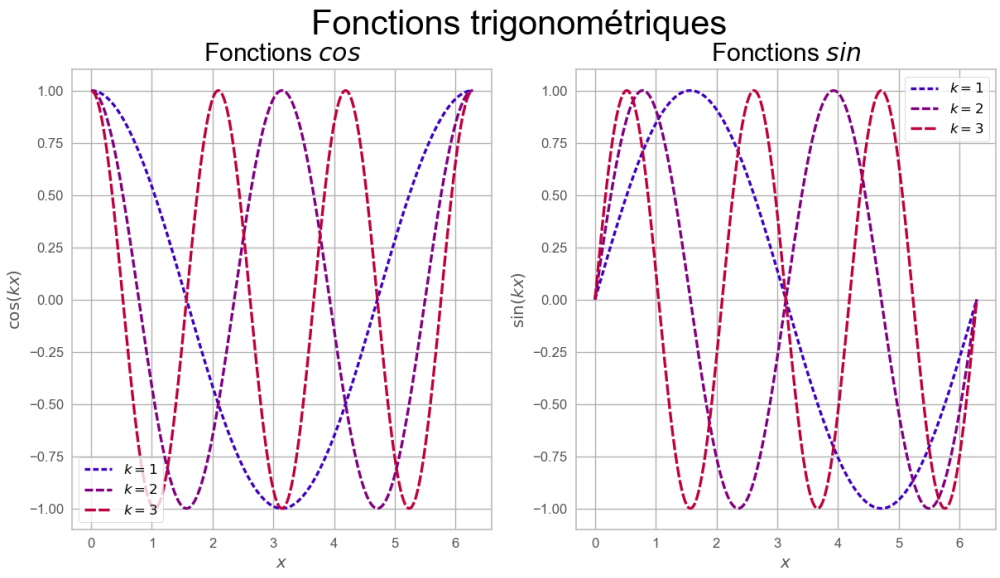
In [39]:
```
def P(x, k):
    return x**k

def Q(x, k):
    return 1-x**k

def test_plot():
    x = np.linspace(0, 1, 100)

    fig = plt.figure(figsize=(12, 6))
    fig.suptitle('Tracé de polynômes', fontsize=25)

    ax_P = fig.add_subplot(1, 2, 1)
    ax_P.grid(False)
    for k in range(6):
        ax_P.plot(
            x, P(x, k),
            label=f'$k={k}$',
            color=(k/6, 0, 1-k/6),
            linewidth=2,
            # color='red',
            linestyle=(0, (k+1, 1))
        )
    ax_P.set_title("$P_k(X)=X^k$", fontsize=18)
    ax_P.legend(fontsize=15)
    ax_P.set_xlabel("$x$")
    ax_P.set_ylabel("$P_k(x)$")

    ax_Q = fig.add_subplot(1, 2, 2)
    ax_P.grid(True)
    for k in range(4):
        ax_Q.plot(
            x, Q(x, k),
            label=f'$k={k}$',
            color=(k/4, 0, 1-k/4), linewidth=2,
            linestyle=(0, (k+1,1))
        )
    ax_Q.set_title("polynôme: $Q_k(X)=1-X^k$", fontsize=18)
    ax_Q.legend(fontsize=15)
    ax_Q.set_xlabel("$x$")
    ax_Q.set_ylabel("$Q_k(x)$")

test_plot()
```



Question

- Proposez une fonction `plot_trigo` qui trace dans deux sous-figures les fonctions $x \mapsto \cos(kx)$ (dans la première sous-figure) et $x \mapsto \sin(kx)$ (dans la seconde sous-figure) sur l'intervalle $[0, 2\pi]$, pour $k \in [1, 2, 3]$.
- Ajoutez des titres et des légendes ainsi que des labels aux axes.
- Modifiez comme vous voulez l'épaisseur des traits, le style des lignes et les couleurs.

In [10]:
```
def plot_trigo():
    x = np.linspace(0, 2*np.pi, 1025)

    fig = plt.figure(figsize=(12, 6))
    fig.suptitle("Fonctions trigonométriques", fontsize=25)

    ax_C = fig.add_subplot(1, 2, 1)
    for k in range(1, 4):
        ax_C.plot(
            x, np.cos(k*x),
            label=r'$k={}$'.format(k),
            color=(k/4, 0, 1-k/4), linewidth=2,
            linestyle=(0, (k+1,1))
        )
    ax_C.set_title(r'Fonctions $cos$', fontsize=18)
    ax_C.legend(loc='lower left')
    ax_C.set_xlabel(r"$x$")
    ax_C.set_ylabel(r"$\cos(kx)$")

    ax_S = fig.add_subplot(1, 2, 2)
    for k in range(1, 4):
        ax_S.plot(
            x, np.sin(k*x),
            label=r'$k={}$'.format(k),
            color=(k/4, 0, 1-k/4), linewidth=2,
            linestyle=(0, (k+1,1))
        )
    ax_S.set_title(r'Fonctions $sin$', fontsize=18)
    ax_S.legend()
    ax_S.set_xlabel(r"$x$")
    ax_S.set_ylabel(r"$\sin(kx)$")

plot_trigo()
```

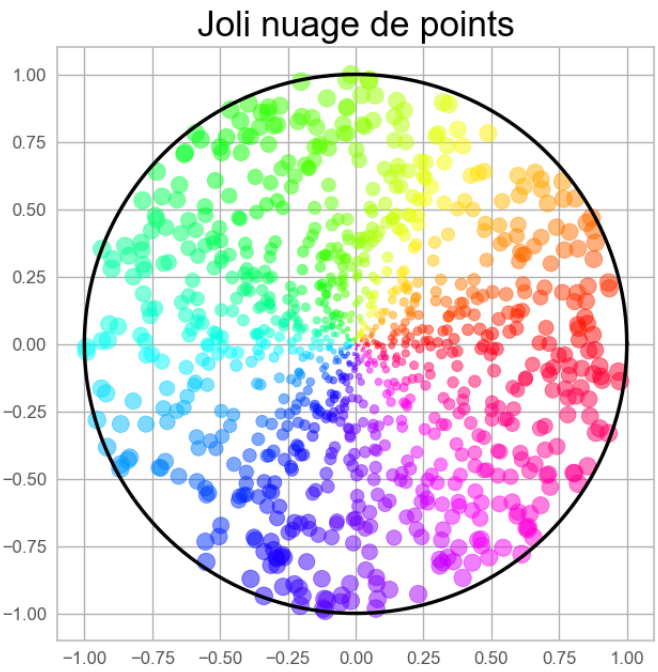


Un exemple de nuage de points

La commande `scatter` permet de tracer des points sans les relier par des lignes. Cela permet de représenter des nuages de points.

Exécutez la cellule suivante pour l'exemple.

In [18]:
```
nb_points = 1000
theta = 2*np.pi*np.random.rand(nb_points)
r = np.random.rand(nb_points)
x, y = r*np.cos(theta), r*np.sin(theta)
tt = np.linspace(0, 2*np.pi, 1000)
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(
    x, y,
    marker='o',
    c=theta,
    s=100*r,
    cmap='hsv', alpha=0.5
)
ax.plot(np.cos(tt), np.sin(tt), color='black', linewidth=2)
ax.set_title("Joli nuage de points", fontsize=20)
```



Out[18]: Text(0.5, 1.0, 'Joli nuage de points')

Méthode des rectangles pour le calcul approché d'intégrales

La méthode des rectangles est une méthode de calcul approché d'intégrales. Vous devez la connaître puisqu'elle est utilisée pour définir les intégrales de Riemann pour les fonctions régulières.

- Définissez une figure `fig` de taille $(12, 12)$. Cette figure sera découpée en $2x2$ sous-figures à l'aide de la commande `add_subplot(2, 2, k)` où `k` est le numéro de la sous-figure entre 1 et 4.
- Dans la première sous-figure, tracez la fonction ainsi que l'aire sous la courbe à l'aide de la fonction `plot_exact` .
- En vous inspirant de la fonction `plot_exact` , complétez la fonction `plot_rect` afin que :
  - elle trace également la fonction f
  - elle remplisse des rectangles définis par les sommets $(x_k, 0)$, $(x_k, f(x_k))$, $(x_{k+1}, f(x_k))$ et $(x_{k+1}, 0)$ pour $0 \leq k \leq N-1$, avec $x_k = a + k(b-a)/N$, $0 \leq k \leq N$.
- Dans les sous-figures 2, 3 et 4, ajoutez le tracé obtenu par la fonction `plot_rect` en prenant $N \in \{3, 8, 20\}$.
- (bonus) *ajoutez en rouge dans la fonction `plot_rect` la ligne brisée qui délimite l'aire approchée par la méthode des rectangles.*

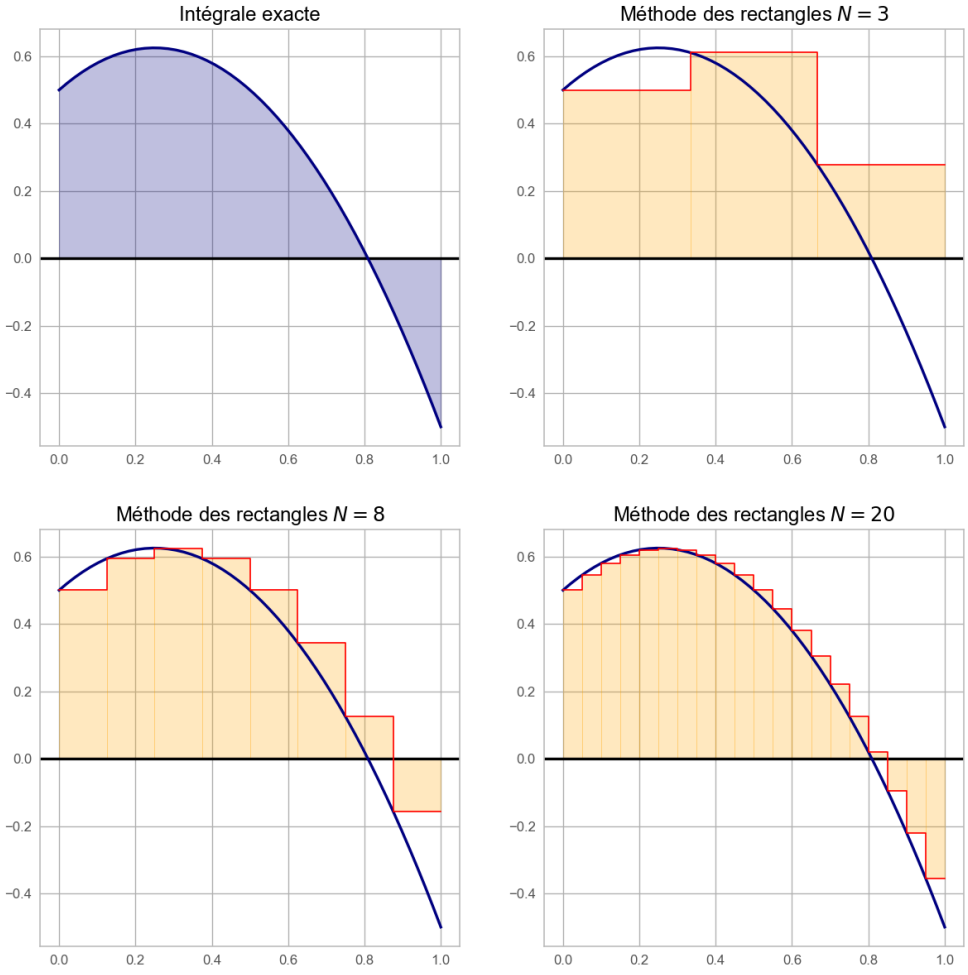
Vous pourrez obtenir la figure suivante à la fin.

methode_des_rectangles

In [19]:
```
def f(x):
    """La fonction dont on veut calculer l'intégrale (aire sous la courbe)"""
    return .5 + x -2*x**2

def plot_exact(ax, a, b):
    """
    Trace la fonction ainsi que l'aire sous la courbe

    Parameters
    ----------

    ax: matplotlib axes
        le calque où la figure doit être tracée

    a: double
        borne inférieure du tracé

    b: double
        borne supérieure du tracé
```

```
    """
    xx = np.linspace(a, b)
    ax.axhline(c='black', lw=2)
    ax.plot(xx, f(xx), lw=2, c='navy')
    ax.fill_between(xx, f(xx), color='navy', alpha=0.25)
    ax.set_title("Intégrale exacte")

def plot_rect(ax, N, a, b):
    """
    Trace la fonction ainsi que l'aire sous la courbe
    approchée par la méthode des rectangles à gauche
    avec N points

    Parameters
    ----------

    ax: matplotlib axes
        le calque où la figure doit être tracée

    N: int
        nombre de points pour la méthode des rectangles

    a: double
        borne inférieure du tracé

    b: double
        borne supérieure du tracé
    """
    xx = np.linspace(a, b)
    ax.axhline(c='black', lw=2)
    ax.plot(xx, f(xx), lw=2, c='navy')
    for k in range(N):
        xl = a + k*(b-a)/N
        xr = a + (k+1)*(b-a)/N
        ax.fill_between(
            [xl, xr],
            [f(xl), f(xl)],
            color='orange', alpha=0.25
        )
    xrect = np.repeat(np.linspace(a, b, N+1), 2)
    yrect = f(xrect)
    ax.plot(xrect[1:-1], yrect[:-2], color='red')
    ax.set_title(f"Méthode des rectangles $N={N}$")
```

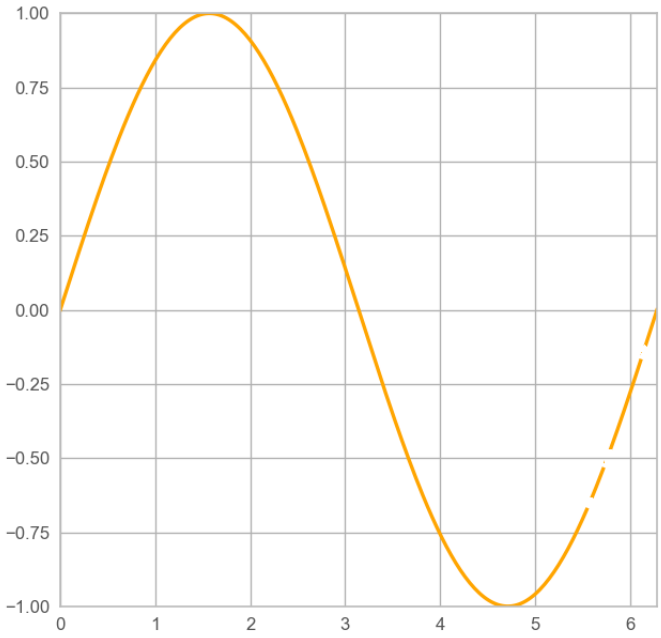
In [20]:
```
a, b = 0, 1
fig = plt.figure(figsize=(12, 12))
plot_exact(fig.add_subplot(2, 2, 1), a, b)
for k, N in enumerate([3, 8, 20]):
    plot_rect(fig.add_subplot(2, 2, k+2), N, 0, 1)
fig.savefig('meth_rect.png')
```



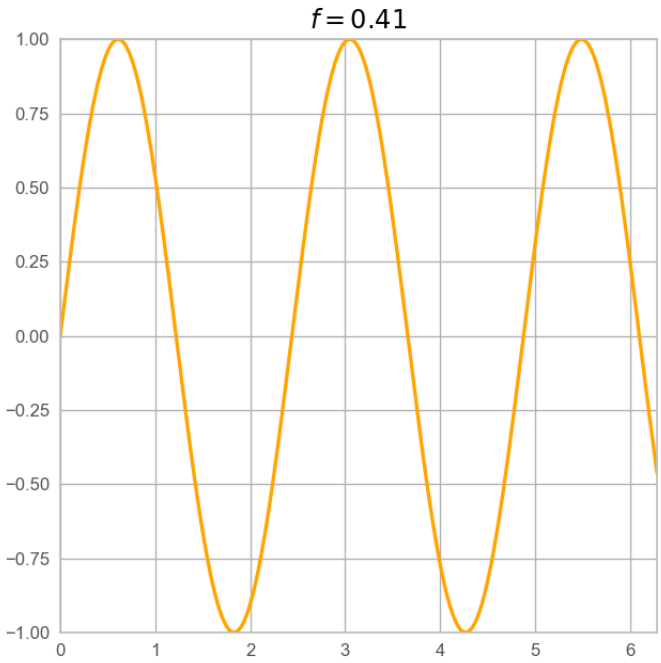
Animation et widgets

`Matplotlib` est un module qui permet de faire de très joli chose comme les deux exemples ci-dessous. On ne vous demandera pas d'être capable de le faire par vous-même...

In [9]:
```
from matplotlib.animation import FuncAnimation

fig = plt.figure(figsize=(6, 6))
fig.clf()

ax = fig.add_subplot(1, 1, 1)
xdata = np.linspace(0, 2*np.pi, 256)
ydata = np.sin(xdata)
ax.set_xlim(0, 2*np.pi)
ax.set_ylim(-1, 1)
ln = ax.plot([], [], linewidth=2, color='orange')[0]

def update(frame):
    if frame < xdata.size+1:
        ln.set_data(xdata[:frame], ydata[:frame])
    return ln,

ani = FuncAnimation(
    fig, update,
    blit=False, interval=20
)
```



In [10]:
```
import ipywidgets as widgets

def phi(t, f):
    return np.sin(2*np.pi*f*t)

t= np.linspace(0, 2*np.pi, 10000)
f = 1
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
line, = ax.plot([], [], color='orange', linewidth=2)
ax.set_title(f"$f={f}$")
ax.set_xlim(0, 2*np.pi)
ax.set_ylim(-1, 1)

def update(f, color):
    line.set_data(t, phi(t, f))
    line.set_color(color)
    ax.set_title(f"$f={f}$")
    plt.show()

widgets.interact(
    update,
    f=widgets.FloatSlider(
        value=1,
        min=0, max=2, step=.01,
        continuous_update=True,
        description='fréquence',
    ),
    color=widgets.Dropdown(
        options=['navy', 'orange', 'brown'],
        value='navy',
        description="couleur"
    )
)
```



Out[10]: <function __main__.update(f, color)>

In [ ]: