Analysons spectralement des fichiers numériques enregistrant une note jouée par un instrument de musique (flûte, orgue, piano).

Vous pouvez les écouter si votre ordinateur possède un logiciel adapté à sa lecture (par exemple itunes).

Les objectifs sont :

(1) Importer le fichier numérique d'extension .wav

(2) Calculer la transformée de Fourier discrète de ce fichier. Visualiser son spectre d'amplitude.

(3) Identifier sa fondamentale donc la note jouée par l'instrument.

Entrer un fichier numérique de format .wav

La bibliothèque scipy.io dispose de nombreuses fonctions permettant de lire et d'écrire des données dans divers formats de fichiers (io pour Input Output).

Pour un fichier son de format Wav, on utilise le module scipy.io.wavfile. La documentation indique que :

$\boxed{\text{read}}$ ouvre un fichier wav $\boxed{\text{write}}$ écrit un tableau numpy sous forme de fichier wav

$\boxed{\text{rate}}$ est le taux d'échantillonnage du fichier WAV $\boxed{\text{data}}$ est un tableau de données

In [36]:
```
import scipy.io.wavfile as wave

rate, data = wave.read('flute.wav') # modifier le chemin d'accès si le fichier n'est pas
```

Pour lire et exploiter le fichier importé, on importe les bibliothèques classiques.

In [37]:
```
import numpy as np
import matplotlib.pyplot as plt

print("La fréquence d'échantillonnage est rate=", rate)
print("La taille du tableau de données est data=",np.size(data))
```

```
La fréquence d'échantillonnage est rate= 48000
La taille du tableau de données est data= 287522
```

Les sons audibles les plus aigus sont à 20000 Hz. Le critère de Shannon impose une fréquence d'échantillonage à 2*20000 au moins.

L'échantillonnage des fichiers audio à 48000 Hz est judicieuse.

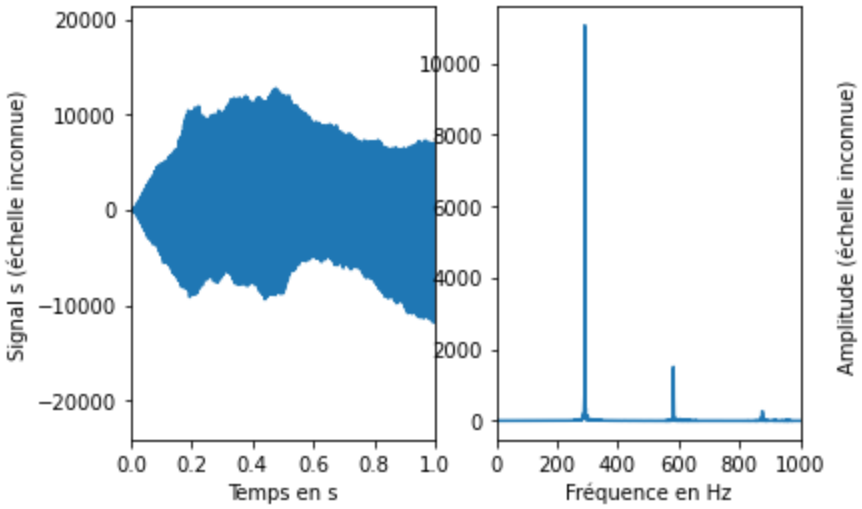
Représenter temporellement et fréquentiellement un signal

On utilise les outils numériques I et VI.

Par simplicité, on garde le nombre d'échantillons du fichier original : $N = 287522$.

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt

##############################################################################

# Définition du signal et des paramètres d'échantillonnage

##############################################################################

fe=rate
Te=1/fe
N=np.size(data)     # donc ici 287522

s=data[0:N]
##############################################################################

# Représentation temporelle

##############################################################################

plt.subplot(121)        #  premier élément d'une matrice 1 ligne, 2 colonnes

t=np.linspace(0,N*Te,N)


plt.plot(t,s,"-")

plt.xlim(0,1)         # Ecoute sur une durée à choisir
plt.xlabel("Temps en s")
plt.ylabel("Signal s (échelle inconnue)")

##############################################################################

# Représentation fréquentielle

##############################################################################


plt.subplot(122)              # second élément d'une matrice 1 ligne, 2 colonnes

hat_s=np.fft.rfft(s)

A_s=2*abs(hat_s)/N                    # Amplitude des sinusoïdes contenues dans le signal
A_s[0]=abs(hat_s[0])/N                # On ne regroupe pas les fréquences 0 et -0 sinon,


f=np.linspace(0,fe/2,N//2+1)
plt.plot(f,A_s)

plt.xlim(0,1000)
plt.xlabel("Fréquence en Hz")
plt.ylabel("Amplitude (échelle inconnue) ",labelpad=-220)    # On a décalé à droite le "l
```

Out[38]:
```
Text(0, 0.5, 'Amplitude (échelle inconnue) ')
```



Type des données

On observe que les nombres associés au fichier numérique $s(t)$ sont compris entre -12000 et +12000.

Evidemment, il ne s'agit pas de nombres en Volt.

L'instruction suivante informe sur le type d'un élément du fichier $data$.

In [39]:
```
type(data[0])
```

Out[39]:
```
numpy.int16
```

Cela signifie que chaque nombre du tableau $data$ est en entier codé sur 16 bit.

Un bit sert au signe, il reste 15 bit pour $2^{15} = 32768$ valeurs.

Un échantillon de s est un entier compris entre -32768 et +32768.

C'est cohérent avec la représentation de $s(t)$ entre -12000 et +12000.

Rechercher la fréquence fondamentale

À l'aide du tableau ci-dessous, (3ème gamme de do centrée sur La_3 que les anglo-saxons appellent A_4), reconnaître la note jouée:

note	DO	Ré	MI	FA	SOL	LA	SI	DO	Ré
Fréquence (Hz)	261,6	293,7	329,6	349,2	392,0	440	493,9	523,2	587,3

Question 1

Quelle est la note jouée par la flûte ?

In [40]:
```
y=A_s                          # Tableau des ordonnées du spectre

m = 0                          # Initialisation du max de l'ordonnée
for i in range(N//2):          # Recherche sur les fréquences audibles (fe/2=48000/2=24000)
    if abs(y[i]) > m:
```

```
        m, fmax = abs(y[i]), f[i]

print("La fréquence fondamentale est",round(fmax,1), "Hz.")    # On ne garde qu'une décim
```

```
La fréquence fondamentale est 291.3 Hz.
```

La note jouée par la flûte est donc Ré (indice 3).

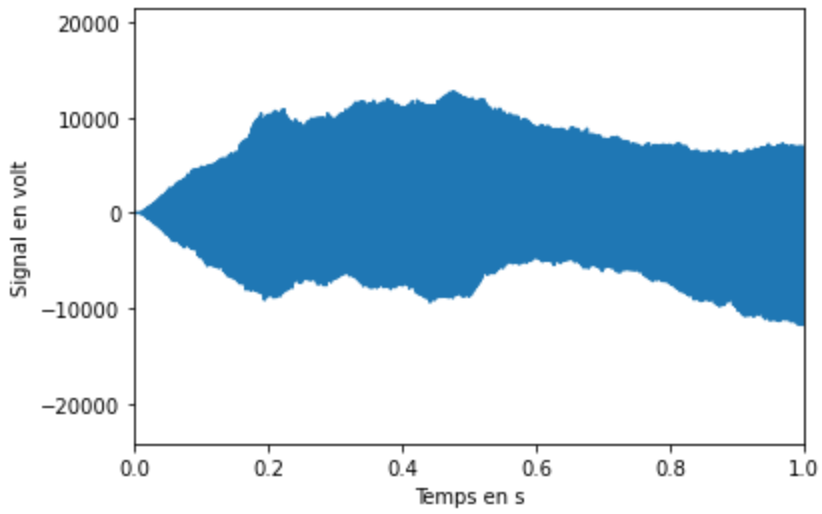
Transformée de Fourier inverse ifft

Cette transformation permet de passer du domaine fréquentiel au domaine temporel : appliquée à $\hat{s}(f)$ on obtient $s(t)$.

Testons cette propriété sur la transformée de Fourier du signal numérique de la flûte.

In [41]:
```
# Représentation temporelle

s1=np.fft.irfft(np.fft.rfft(s))    # Tableau de réels

t=np.linspace(0,N*Te,N)
plt.plot(t,s1)

plt.xlabel("Temps en s")
plt.ylabel("Signal en volt")
plt.xlim(0,1)

print(type(s1[0]))
```

```
<class 'numpy.float64'>
```



Constat : Visuellement le signal temporel $s_1(t)$ issu de la double transformation est identique au tableau $s(t)$ (et compris entre 12000 et -12000).

Sortir un fichier numérique pour l'écouter

Il faut convertir le tableau de réels $s_1(t)$ en entiers codés sur 16 bits (comme vu en section 3 et cf documentation de scipy.io).

Puis écrire un fichier wav. puis écouter avec un logiciel d'ordinateur (comme itunes).

In [42]:
```
s1_audio=s1.astype(np.int16)                    # On tranforme s1 en entiers codés sur 16 b
```

```
wave.write('fluteCANA.wav',rate,s1_audio)     # On peut lire le fichier 'fluteCANA' avec
```

Constat : le son de la flûte est identique après les 4 opérations successives : importation-rfft-irfft-exportation.

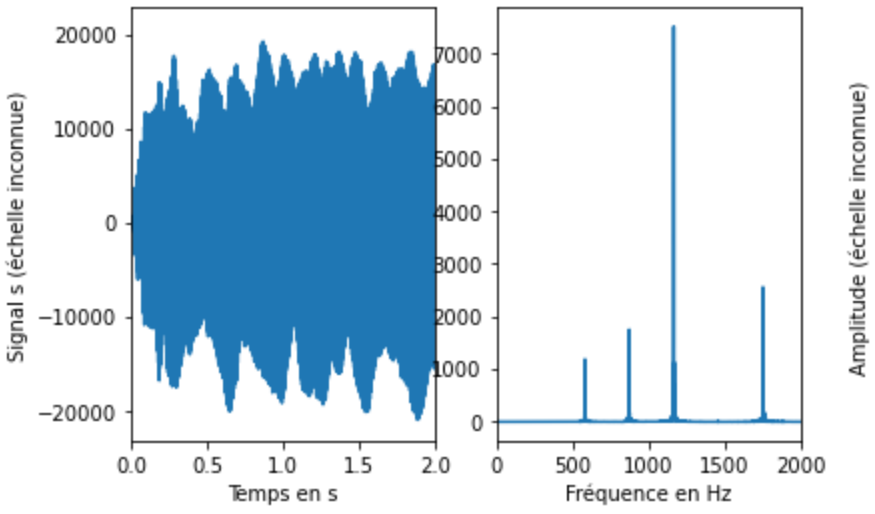
Analyse d'un son d'orgue

On utilise le même code que pour la flûte.

In [57]:
```
##############################################################################

# Entrée du fichier son et informations sur le fichier

##############################################################################

import scipy.io.wavfile as wave


rate, data = wave.read('orgue.wav')      # Enregistrement d'un son d'orgue

# Information sur 'rate'                        # rate = taux d'échantillonnage

print("La fréquence d'échantillonnage est rate=", rate)

# Information sur 'data'

print("Le type des données est ",type(data))
print("La forme des données est ",data.shape)
print("La taille des données est data=",np.size(data) )

##############################################################################

# Définition du signal et des paramètres d'échantillonnage

##############################################################################


import numpy as np
import matplotlib.pyplot as plt

fe=rate
Te=1/fe

s=data[0:N]                              # Tableau de N éléments (nouvelle valeur de N)


##############################################################################

# Représentation temporelle

##############################################################################

plt.subplot(121)       #  premier élément d'une matrice 1 ligne, 2 colonnes

t=np.linspace(0,N*Te,N)

plt.plot(t,s,"-")

plt.xlim(0,2)         # Ecoute sur une durée à choisir
plt.xlabel("Temps en s")
plt.ylabel("Signal s (échelle inconnue)")
```

```
##############################################################################

# Représentation fréquentielle

##############################################################################


plt.subplot(122)                        # second élément d'une matrice 1 ligne, 2 colonnes

hat_s=np.fft.rfft(s)

A_s=2*abs(hat_s)/N                      # Amplitude des sinusoïdes contenues dans le signal
A_s[0]=abs(hat_s[0])/N                  # On ne regroupe pas les fréquences 0 et -0 sinon,

f=np.linspace(0,fe/2,N//2+1)
plt.plot(f,A_s)

plt.xlim(0,2000)
plt.xlabel("Fréquence en Hz")
plt.ylabel("Amplitude (échelle inconnue) ",labelpad=-220)    # On a décalé à droite le "l

##############################################################################

# Calcul de la fréquence fondamentale

##############################################################################

# Pour trouver la fondamentale, il faut se limiter au voisinage du premier pic.
# L'indice k correspondant à 500 Hz est donné par k*fe/N=500

k=round(600*N/fe)                               # Il faut modifier le code de la flûte
#print("La valeur du range est k=",k)

y=A_s    # Tableau des ordonnées du spectre

m = 0
for i in range(k):        # Maximum horizontal à ajuster pour avoir le pic fondamental
    if abs(y[i]) > m:
        m, fmax = abs(y[i]), f[i]

print("La fréquence fondamentale est",round(fmax,1),"Hz.")    # On ne garde qu'une décima
```

```
La fréquence d'échantillonnage est rate= 48000
Le type des données est  <class 'numpy.ndarray'>
La forme des données est  (277016,)
La taille des données est data= 277016
La fréquence fondamentale est 583.6 Hz.
```



La note jouée à l'orgue est un Ré4. Sa fréquence est le double de celle de la flûte.

NB : L'intervalle qui sépare ces deux notes est appellée $octave$.

Analyse du son stéréo d'une note de piano

Source du fichier audio : University of Iowa Electronic music studios

https://theremin.music.uiowa.edu/MISpiano.html

L'écoute du fichier audio (avec itunes) informe que le fichier dure 30 s.

In [53]:
```
################################################################################

# Entrée du fichier son et informations sur le fichier

################################################################################

import scipy.io.wavfile as wave


rate, data = wave.read('Piano.mf.D4.wav')      # Enregistrement sur piano Steinway de la

# Information sur 'rate'                       # rate = taux d'échantillonnage

print("La fréquence d'échantillonnage est rate=", rate)

# Information sur 'data'

print("Le type des données est ",type(data))
print("La forme des données est ",data.shape)
print("La taille des données est data=",np.size(data) )
print("Le type d'une des données est ",type(data[0,0]))
```

```
La fréquence d'échantillonnage est rate= 44100
Le type des données est  <class 'numpy.ndarray'>
La forme des données est  (1299317, 2)
La taille des données est data= 2598634
Le type d'une des données est  <class 'numpy.int16'>
```

'data' est un signal échantillonné à $f_e = 44,1$ kHz.

C'est un tableau de dimension 2 : 1 299 317 lignes et 2 colonnes donc 1 299 317*2=2598634 éléments.

Chaque élément est un entier codé sur 16 bits.

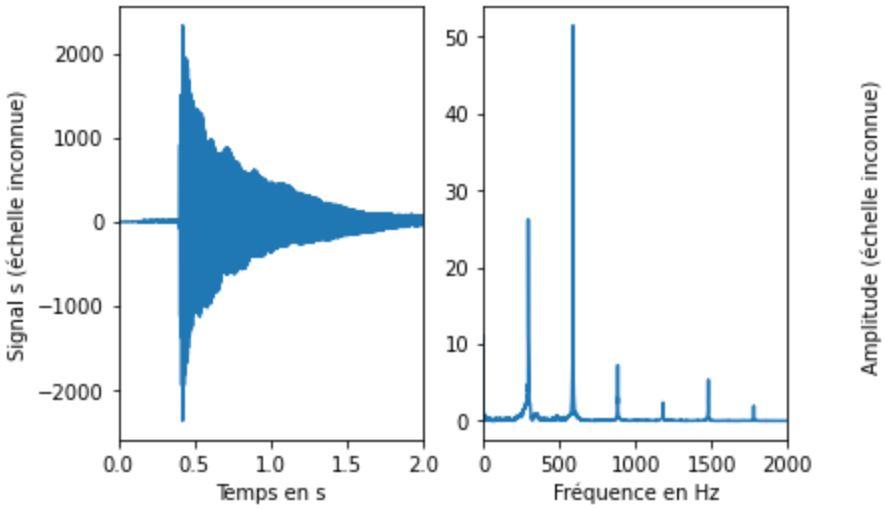
Avec la documentation wav, on comprend qu'il s'agit d'un son stéréo (2 canaux=2 colonnes).

Analyse du premier canal du son stéréo

In [44]:
```
# On n'étudie que le premier canal :

N=np.size(data)//2                # Nombre d'échantillons dans chaque colonne
data=data[0:N,0]                  # Choix des éléments des lignes 0 à N-1, colonne 1

# On réduit le fichier à 10 s (son audible sur 10 s) :

N=N//3
```

In [45]:
```
import numpy as np
import matplotlib.pyplot as plt

##############################################################################

# Définition du signal et des paramètres d'échantillonnage

##############################################################################

fe=rate
Te=1/fe

s=data[0:N]                              # Tableau de N éléments (nouvelle valeur de N)


##############################################################################

# Représentation temporelle

##############################################################################

plt.subplot(121)        #  premier élément d'une matrice 1 ligne, 2 colonnes

t=np.linspace(0,N*Te,N)

plt.plot(t,s,"-")

plt.xlim(0,2)         # Ecoute sur une durée à choisir
plt.xlabel("Temps en s")
plt.ylabel("Signal s (échelle inconnue)")

##############################################################################

# Représentation fréquentielle

##############################################################################


plt.subplot(122)              # second élément d'une matrice 1 ligne, 2 colonnes

hat_s=np.fft.rfft(s)

A_s=2*abs(hat_s)/N                     # Amplitude des sinusoïdes contenues dans le signal
A_s[0]=abs(hat_s[0])/N                 # On ne regroupe pas les fréquences 0 et -0 sinon,

f=np.linspace(0,fe/2,N//2+1)
plt.plot(f,A_s)

plt.xlim(0,2000)
plt.xlabel("Fréquence en Hz")
plt.ylabel("Amplitude (échelle inconnue) ",labelpad=-220)    # On a décalé à droite le "l
```

Out[45]:
```
Text(0, 0.5, 'Amplitude (échelle inconnue) ')
```



In [46]:
```
# Pour trouver la fondamentale, il faut se limiter au voisinage du premier pic.
# L'indice k correspondant à 500 Hz est donné par k*fe/N=500

k=round(500*N/fe)
print("L'indice de l'échantillon correspondant à 500 Hz est k=",k)

y=A_s    # Tableau des ordonnées du spectre

m = 0
for i in range(k):        # Maximum horizontal à ajuster pour avoir le pic fondamental
    if abs(y[i]) > m:
        m, fmax = abs(y[i]), f[i]

print("La fréquence fondamentale est",round(fmax,1),"Hz.")    # On ne garde qu'une décima
```

```
L'indice de l'échantillon correspondant à 500 Hz est k= 4910
La fréquence fondamentale est 294.0 Hz.
```

- La note jouée par le piano est aussi un Ré (indice 3).

C'est la même note que celle jouée par la flûte mais le *timbre* est différent car les harmoniques diffèrent en amplitudes.

On peut remarquer que l'harmonique de rang 2 a une amplitude plus grande que celle de rang 1 (rare).

- Recommençons la même expérience mais avec le canal 2 du son enregistré.

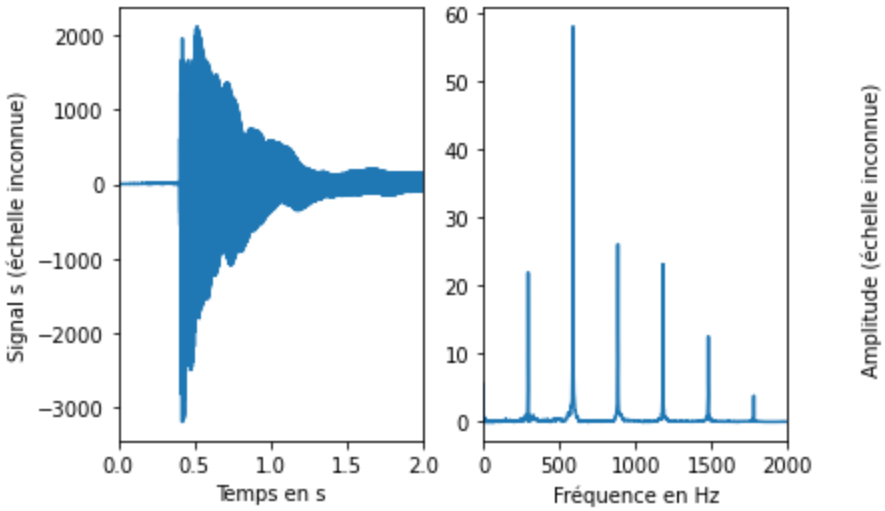
Analyse du second canal stéréo

In [47]:
```
##############################################################################

# Entrée du fichier son et informations sur le fichier

##############################################################################

import scipy.io.wavfile as wave


rate, data = wave.read('Piano.mf.D4.wav')       # Enregistrement sur piano Steinway de la

# Information sur 'rate'                        # rate = taux d'échantillonnage

print("La fréquence d'échantillonnage est rate=", rate)
```

```
# Information sur 'data'

print("Le type des données est ",type(data))
print("La forme des données est ",data.shape)
print("La taille des données est data=",np.size(data) )


#############################################################################

# Extraction du second canal et réduction du nombre de données


#############################################################################

N=np.size(data)//2                # Nombre d'échantilons dans chaque colonne
data=data[0:N,1]                  # Choix des éléments des lignes 0 à N-1, colonne 2


# On réduit le fichier à 10 s (son audible sur 10 s) :

N=N//3

#############################################################################

# Définition du signal et des paramètres d'échantillonnage

#############################################################################


import numpy as np
import matplotlib.pyplot as plt

fe=rate
Te=1/fe

s=data[0:N]                               # Tableau de N éléments (nouvelle valeur de N)


#############################################################################

# Représentation temporelle

#############################################################################

plt.subplot(121)        #  premier élément d'une matrice 1 ligne, 2 colonnes

t=np.linspace(0,N*Te,N)

plt.plot(t,s,"-")

plt.xlim(0,2)         # Ecoute sur une durée à choisir
plt.xlabel("Temps en s")
plt.ylabel("Signal s (échelle inconnue)")

#############################################################################

# Représentation fréquentielle

#############################################################################


plt.subplot(122)                      # second élément d'une matrice 1 ligne, 2 colonnes

hat_s=np.fft.rfft(s)

A_s=2*abs(hat_s)/N                    # Amplitude des sinusoïdes contenues dans le signal
```

```
A_s[0]=abs(hat_s[0])/N                # On ne regroupe pas les fréquences 0 et -0 sinon,

f=np.linspace(0,fe/2,N//2+1)
plt.plot(f,A_s)

plt.xlim(0,2000)
plt.xlabel("Fréquence en Hz")
plt.ylabel("Amplitude (échelle inconnue) ",labelpad=-220)    # On a décalé à droite le "l

##############################################################################

# Calcul de la fréquence fondamentale

##############################################################################

# Pour trouver la fondamentale, il faut se limiter au voisinage du premier pic.
# L'indice k correspondant à 500 Hz est donné par k*fe/N=500

k=round(500*N/fe)
#print("La valeur du range est k=",k)

y=A_s     # Tableau des ordonnées du spectre

m = 0
for i in range(k):        # Maximum horizontal à ajuster pour avoir le pic fondamental
    if abs(y[i]) > m:
        m, fmax = abs(y[i]), f[i]

print("La fréquence fondamentale est",round(fmax,1),"Hz.")    # On ne garde qu'une décima
```

```
La fréquence d'échantillonnage est rate= 44100
Le type des données est  <class 'numpy.ndarray'>
La forme des données est  (1299317, 2)
La taille des données est data= 2598634
La fréquence fondamentale est 293.9 Hz.
```



Constat :

C'est bien la même note (fondamentale Ré3) mais avec les harmoniques de rang 3, 4 et 5 de plus forte amplitude.

En stéréo, on utilise deux capteurs positionnés en deux lieux, enregistrant chacun un fichier audio.

Avec deux écouteurs (oreille droite et oreille gauche par exemple), la sensation stéréo est plus proche du son entendu en direct.

On peut penser que le second canal a été capturé sur la droite du pianiste (plus de composantes aiguës).

Vérification du diapason à l'aide du fichier note A4

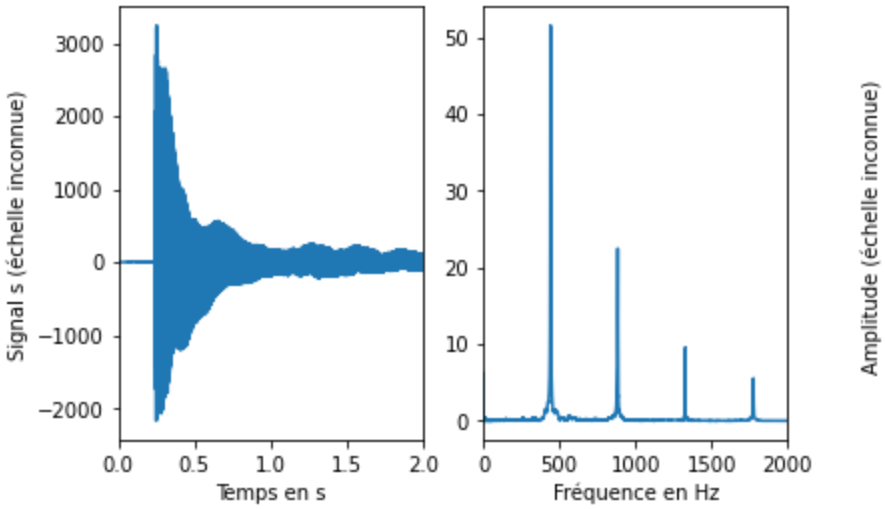
La bijection entre notes et de fréquence fondamentale est relative et repose sur un choix de référence. Le *diapason* définit la fréquence du la3 (= A4).

Usuellement le diapason est à 440 Hz mais peut être à 442 Hz (pour un son plus brillant et moderne) ou 415 Hz (pour un son plus baroque).

In [48]:
```
##############################################################################

# Entrée du fichier son et informations sur le fichier

##############################################################################

import scipy.io.wavfile as wave


rate, data = wave.read('Piano.mf.A4.wav')      # Enregistrement sur piano Steinway de la

# Information sur 'rate'                       # rate = taux d'échantillonnage

print("La fréquence d'échantillonnage est rate=", rate)

# Information sur 'data'

print("Le type des données est ",type(data))
print("La forme des données est ",data.shape)
print("La taille des données est data=",np.size(data) )


##############################################################################

# Extraction du second canal et réduction du nombre de données


##############################################################################

N=np.size(data)//2                # Nombre d'échantilons dans chaque colonne
data=data[0:N,1]                  # Choix des éléments des lignes 0 à N-1, colonne 2


# On réduit le fichier à 10 s (son audible sur 10 s) :

N=N//3

##############################################################################

# Définition du signal et des paramètres d'échantillonnage

##############################################################################


import numpy as np
import matplotlib.pyplot as plt

fe=rate
Te=1/fe
```

```
s=data[0:N]                                  # Tableau de N éléments (nouvelle valeur de N)


##############################################################################

# Représentation temporelle

##############################################################################

plt.subplot(121)        #  premier élément d'une matrice 1 ligne, 2 colonnes

t=np.linspace(0,N*Te,N)

plt.plot(t,s,"-")

plt.xlim(0,2)         # Ecoute sur une durée à choisir
plt.xlabel("Temps en s")
plt.ylabel("Signal s (échelle inconnue)")

##############################################################################

# Représentation fréquentielle

##############################################################################


plt.subplot(122)                        # second élément d'une matrice 1 ligne, 2 colonnes

hat_s=np.fft.rfft(s)

A_s=2*abs(hat_s)/N                      # Amplitude des sinusoïdes contenues dans le signal
A_s[0]=abs(hat_s[0])/N                  # On ne regroupe pas les fréquences 0 et -0 sinon,

f=np.linspace(0,fe/2,N//2+1)
plt.plot(f,A_s)

plt.xlim(0,2000)
plt.xlabel("Fréquence en Hz")
plt.ylabel("Amplitude (échelle inconnue) ",labelpad=-220)    # On a décalé à droite le "l

##############################################################################

# Calcul de la fréquence fondamentale

##############################################################################

# Pour trouver la fondamentale, il faut se limiter au voisinage du premier pic.
# L'indice k correspondant à 500 Hz est donné par k*fe/N=500

k=round(500*N/fe)

y=A_s    # Tableau des ordonnées du spectre

m = 0
for i in range(k):        # Maximum horizontal à ajuster pour avoir le pic fondamental
    if abs(y[i]) > m:
        m, fmax = abs(y[i]), f[i]

print("La fréquence fondamentale est",round(fmax,1),"Hz.")    # On ne garde q'une décimal
```

```
La fréquence d'échantillonnage est rate= 44100
Le type des données est  <class 'numpy.ndarray'>
La forme des données est  (1322313, 2)
La taille des données est data= 2644626
La fréquence fondamentale est 440.7298591 Hz.
```



Constat

Aux incertitudes près, on constate que le piano est accordé avec un diapason à 440 Hz.

Possible évolution dans le temps des amplitudes

La représentation temporelle montre que l'enveloppe du son diminue dans le temps.

Analysons le début du son : écoute sur 0,3s au lieu de 10s.

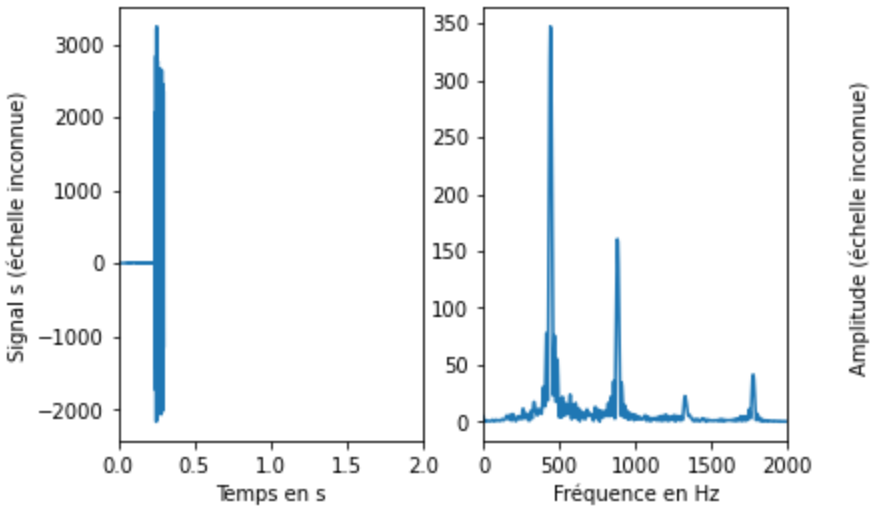
In [49]:

```
################################################################################

# Entrée du fichier son et informations sur le fichier

################################################################################

import scipy.io.wavfile as wave


rate, data = wave.read('Piano.mf.A4.wav')       # Enregistrement sur piano Steinway de la

# Information sur 'rate'                        # rate = taux d'échantillonnage

print("La fréquence d'échantillonnage est rate=", rate)

# Information sur 'data'

print("Le type des données est ",type(data))
print("La forme des données est ",data.shape)
print("La taille des données est data=",np.size(data) )


################################################################################

# Extraction du second canal et réduction du nombre de données

################################################################################

N=np.size(data)//2//100                 # Durée de 30/100=0,3 s environ (attaque de la not

data=data[0:N,1]                    # Choix des éléments des lignes 0 à N-1, colonne 2
```

```
##############################################################################

# Définition du signal et des paramètres d'échantillonnage

##############################################################################


import numpy as np
import matplotlib.pyplot as plt

fe=rate
Te=1/fe

s=data[0:N]                              # Tableau de N éléments (nouvelle valeur de N)


##############################################################################

# Représentation temporelle

##############################################################################

plt.subplot(121)

t=np.linspace(0,N*Te,N)  # Réduction du temps sur la seconde moitié

plt.plot(t,s,"-")

plt.xlim(0,2)         # Ecoute sur une durée à choisir
plt.xlabel("Temps en s")
plt.ylabel("Signal s (échelle inconnue)")

##############################################################################

# Représentation fréquentielle

##############################################################################


plt.subplot(122)                         # second élément d'une matrice 1 ligne, 2 colonnes

hat_s=np.fft.rfft(s)

A_s=2*abs(hat_s)/N                       # Amplitude des sinusoïdes contenues dans le signal
A_s[0]=abs(hat_s[0])/N                   # On ne regroupe pas les fréquences 0 et -0 sinon,

f=np.linspace(0,fe/2,N//2+1)
plt.plot(f,A_s)

plt.xlim(0,2000)
plt.xlabel("Fréquence en Hz")
plt.ylabel("Amplitude (échelle inconnue) ",labelpad=-220)    # On a décalé à droite le "l


##############################################################################

# Calcul de la fréquence fondamentale

##############################################################################

# Pour trouver la fondamentale, il faut se limiter au voisinage du premier pic.
# L'indice k correspondant à 500 Hz est donné par k*fe/N=500

k=round(500*N/fe)

y=A_s     # Tableau des ordonnées du spectre
```

```
m = 0
for i in range(k):        # Maximum horizontal à ajuster pour avoir le pic fondamental
    if abs(y[i]) > m:
        m, fmax = abs(y[i]), f[i]

print("La fréquence fondamentale est",round(fmax,1),"Hz.")    # On ne garde q'une décimal
```

```
La fréquence d'échantillonnage est rate= 44100
Le type des données est  <class 'numpy.ndarray'>
La forme des données est  (1322313, 2)
La taille des données est data= 2644626
La fréquence fondamentale est 440.266223 Hz.
```



On constate qu'au moment de l'attaque de la note, le spectre est différent mais conserve sa fréquence fondamentale.

- La fondamentale a une amplitude 7 fois plus grande que dans l'expérience précédente (son plus fort);
- La troisième harmonique est relativement moins haute.