

1. Effet de la fréquence du signal

1.1. Charger les signaux d'origine et les écouter

```
1 begin
2     chwet_file = joinpath(@__DIR__, "..", "2022", "TP4", "chwet.wav")
3     chwet, fs_chwet = wavread(chwet_file)
4     chwet = chwet[:, 1] # On ne prend qu'un canal
5     println("Nouvelle fréquence d'échantillonnage : $fs_chwet Hz")
6     println("Durée totale du signal : $(length(chwet)/fs_chwet) s")
7 end
```

Nouvelle fréquence d'échantillonnage : 44100.0 Hz
Durée totale du signal : 1.4512472 s

64000

```
1 PortAudioStream() do s
2     write(s, chwet)
3 end
```

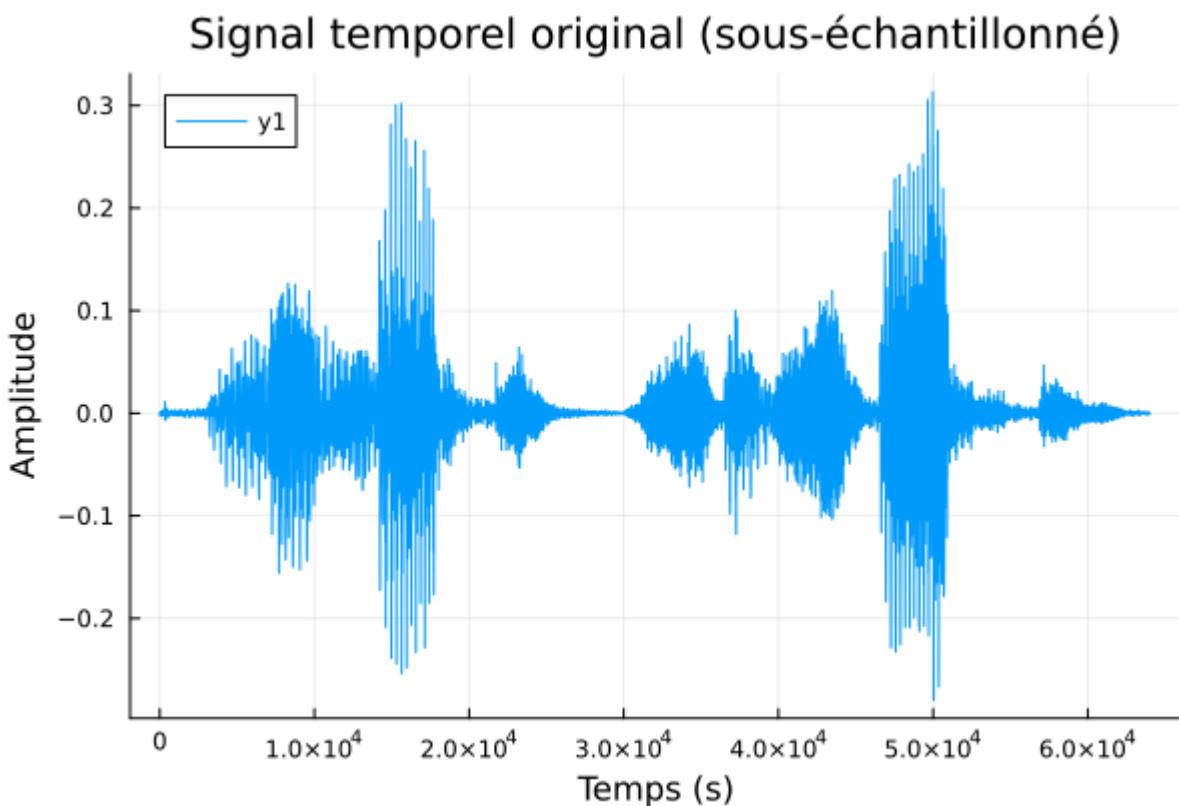
```
1 begin
2     gamme_file = joinpath(@__DIR__, "..", "2022", "TP4", "gamme.wav")
3     gamme, fs_gamme = wavread(gamme_file)
4     gamme = gamme[:, 1] # On ne prend qu'un canal
5     println("Nouvelle fréquence d'échantillonnage : $fs_gamme Hz")
6     println("Durée totale du signal : $(length(gamme)/fs_gamme) s")
7 end
```

Nouvelle fréquence d'échantillonnage : 44100.0 Hz
Durée totale du signal : 5.05034 s

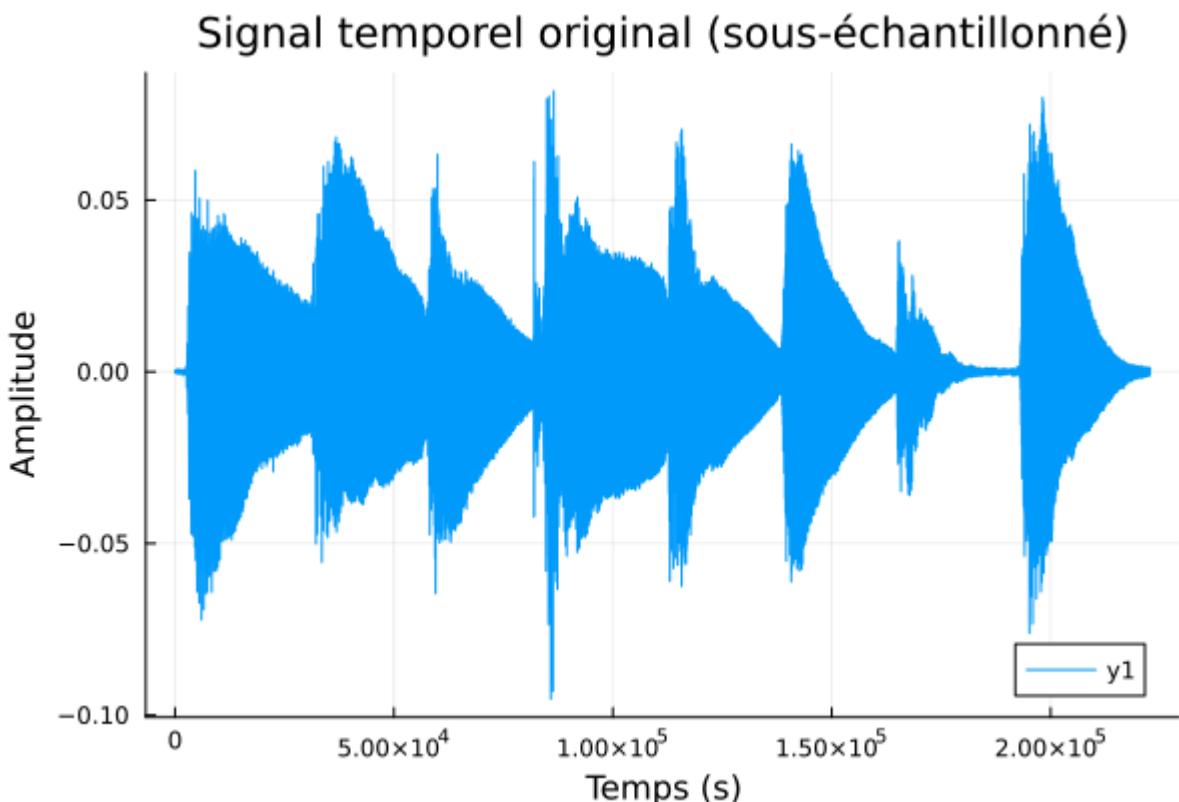
222720

```
1 PortAudioStream() do s
2     write(s, gamme)
3 end
```

1.2. Afficher les signaux d'origine



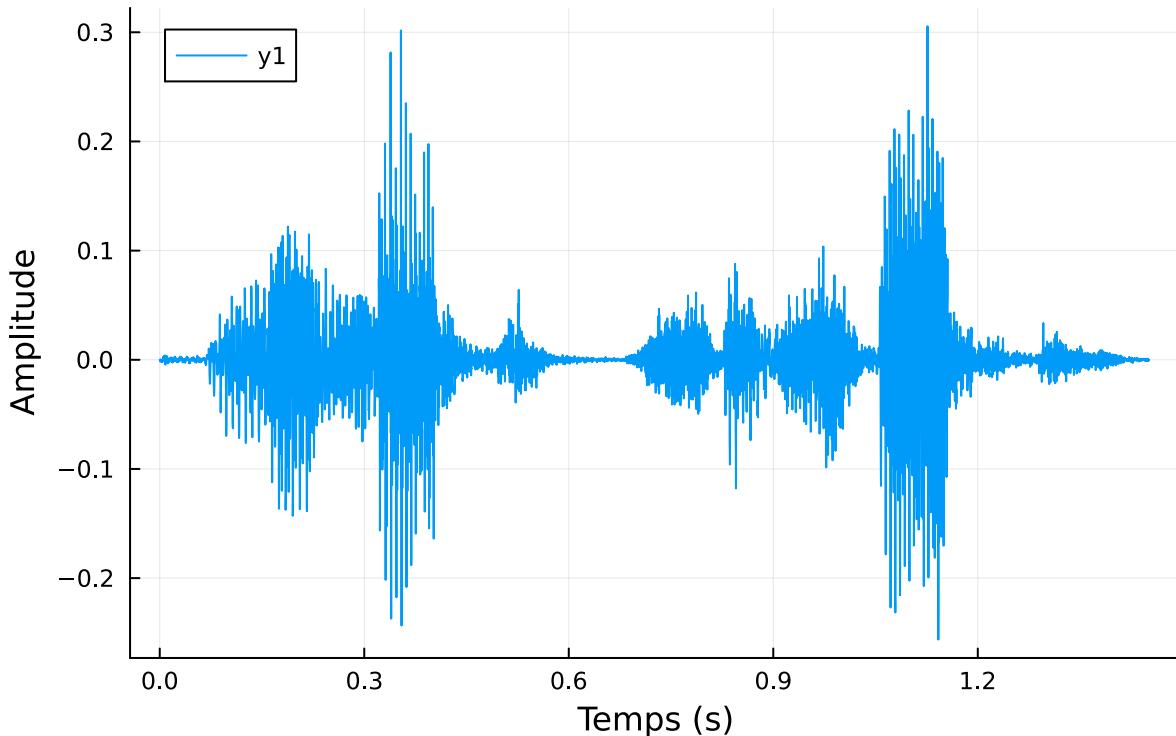
```
1 begin
2     plot(chwet, title="Signal temporel original (sous-échantillonné)",
3         xlabel="Temps (s)", ylabel="Amplitude")
4 end
```



Que remarquez-vous et comment y remédier ? Implémentez la solution ci-dessous

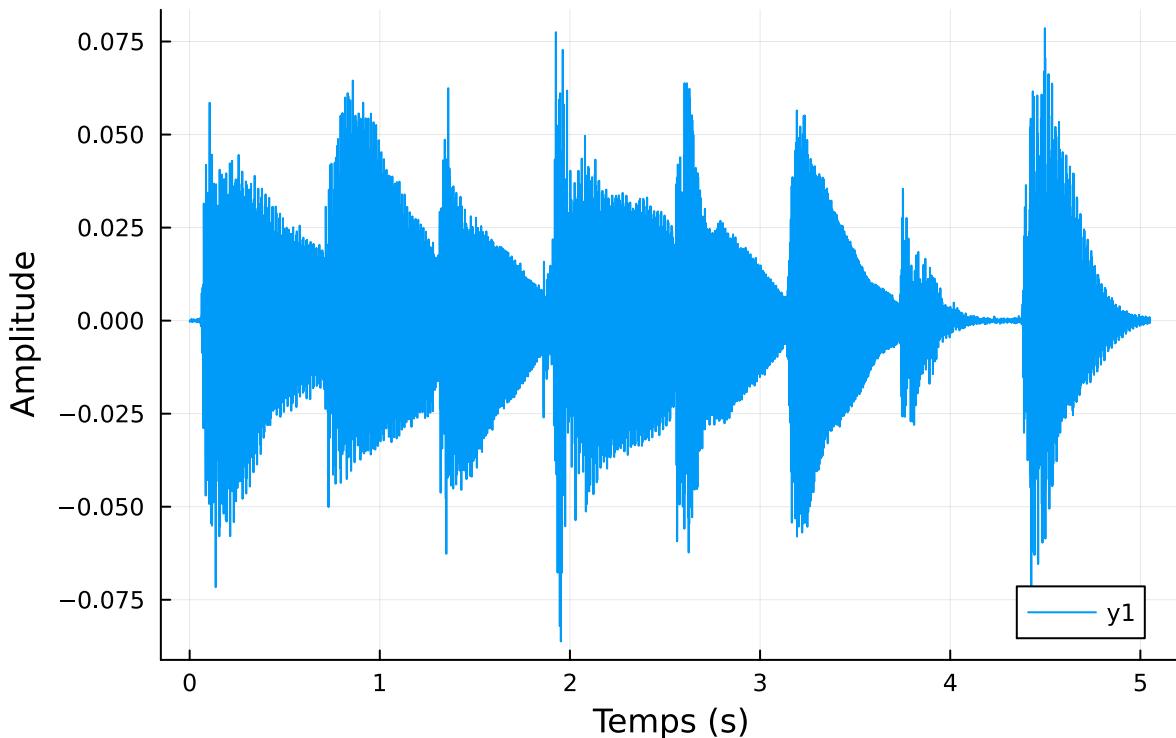
- 1 md"**Que remarquez-vous et comment y remédier ? Implémentez la solution ci-dessous**"
- 2 # To do

Signal temporel original (sous-échantillonné)



```
1 # Echantillonage : prendre 1 échantillon sur 10
2 begin
3     factor_chwet = 10
4     n_chwet = length(chwet)
5     chwet_sub = chwet[1:factor_chwet:n_chwet]
6
7     Δt_chwet = 1 / fs_chwet
8     duration_chwet = n_chwet * Δt_chwet
9     t_chwet_sub = LinRange(0, duration_chwet, length(chwet_sub))
10
11    # Affichage du signal sous-échantillonné
12    plot(t_chwet_sub, chwet_sub, title="Signal temporel original
(sous-échantillonné)", xlabel="Temps (s)", ylabel="Amplitude")
13 end
```

Signal temporel original (sous-échantillonné)



Que remarquez vous en modifiant le sous échantillonage ?

```
1 md"**Que remarquez vous en modifiant le sous échantillonage ?**"
```

1.3. Jouer sur la fréquence des signaux

play_with_modified_frequency (generic function with 1 method)

```
1 # Fonction pour jouer un signal avec une fréquence modifiée
2 function play_with_modified_frequency(signal, fs, factor)
3     new_fs = fs * factor
4     sig = SampleBuf(signal, new_fs)
5     println("Ancienne fréquence d'échantillonnage : $fs Hz")
6     println("Ancienne durée totale du signal : $(length(signal)/fs) s")
7     println("Nouvelle fréquence d'échantillonnage : $new_fs Hz")
8     println("Durée totale du signal : $(length(sig)/new_fs) s")
9     return sig
10 end
```

0:00 / 0:00

chwet_double_freq =

```
1 # Doubler la fréquence
2 chwet_double_freq = play_with_modified_frequency(chwet, fs_chwet, 2)
```

Ancienne fréquence d'échantillonnage : 44100.0 Hz
Ancienne durée totale du signal : 1.4512472 s
Nouvelle fréquence d'échantillonnage : 88200.0 Hz
Durée totale du signal : 0.7256236 s

②

0:00 / 0:02

```
gamme_double_freq =
```

```
1 gamme_double_freq = play_with_modified_frequency(gamme, fs_gamme, 2)
```

```
Ancienne fréquence d'échantillonnage : 44100.0 Hz  
Ancienne durée totale du signal : 5.05034 s  
Nouvelle fréquence d'échantillonnage : 88200.0 Hz  
Durée totale du signal : 2.52517 s
```

(?)

0:00 / 0:02

```
chwet_half_freq =
```

```
1 # Diviser la fréquence par deux  
2 chwet_half_freq = play_with_modified_frequency(chwet, fs_chwet, 0.5)
```

```
Ancienne fréquence d'échantillonnage : 44100.0 Hz  
Ancienne durée totale du signal : 1.4512472 s  
Nouvelle fréquence d'échantillonnage : 22050.0 Hz  
Durée totale du signal : 2.9024943310657596 s
```

(?)

0:00 / 0:10

```
gamme_half_freq =
```

```
1 gamme_half_freq = play_with_modified_frequency(gamme, fs_gamme, 0.5)
```

```
Ancienne fréquence d'échantillonnage : 44100.0 Hz  
Ancienne durée totale du signal : 5.05034 s  
Nouvelle fréquence d'échantillonnage : 22050.0 Hz  
Durée totale du signal : 10.100680272108843 s
```

(?)

Que constatez-vous ?

```
1 md"**Que constatez-vous ?**"  
2 # ...
```

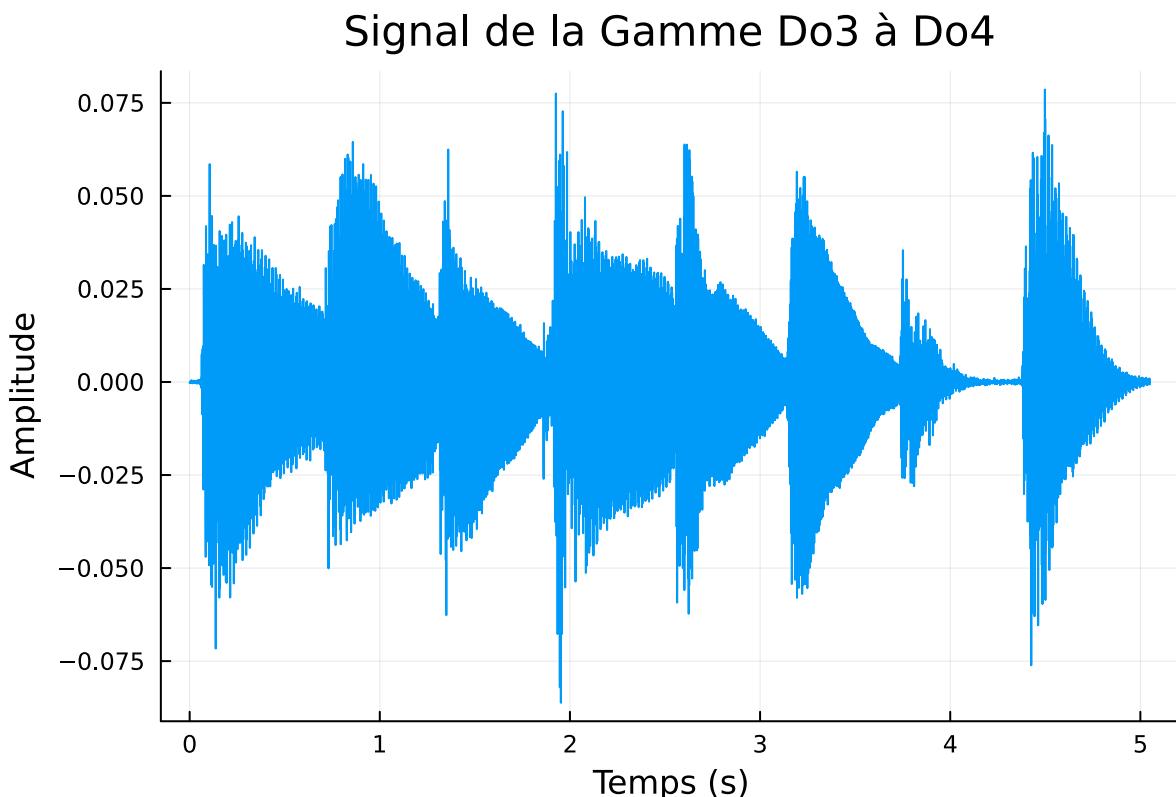
2. Le signal et son spectre

2.1. Afficher le signal

Pour cet exemple, nous allons considérer le signal de la gamme qui va du **Do3** (261,5 Hz) au **Do4** (523 Hz). Pour ce faire, déterminez le vecteur temps nécessaire à afficher le signal complet, puis afficher un graphique ($t, y(t)$) qui affiche le signal $y(t)$ en fonction du temps.

```
time_vector (generic function with 1 method)
```

```
1 # Calcul du vecteur de temps
2 function time_vector(signal, signal_sub, fs)
3     nb_echantillon = length(signal)    # Nombre total d'échantillons dans le signal
4     Δt = 1 / fs
5     duration = nb_echantillon * Δt
6     temps = LinRange(0, duration, length(signal_sub))
7     return temps
8 end
```



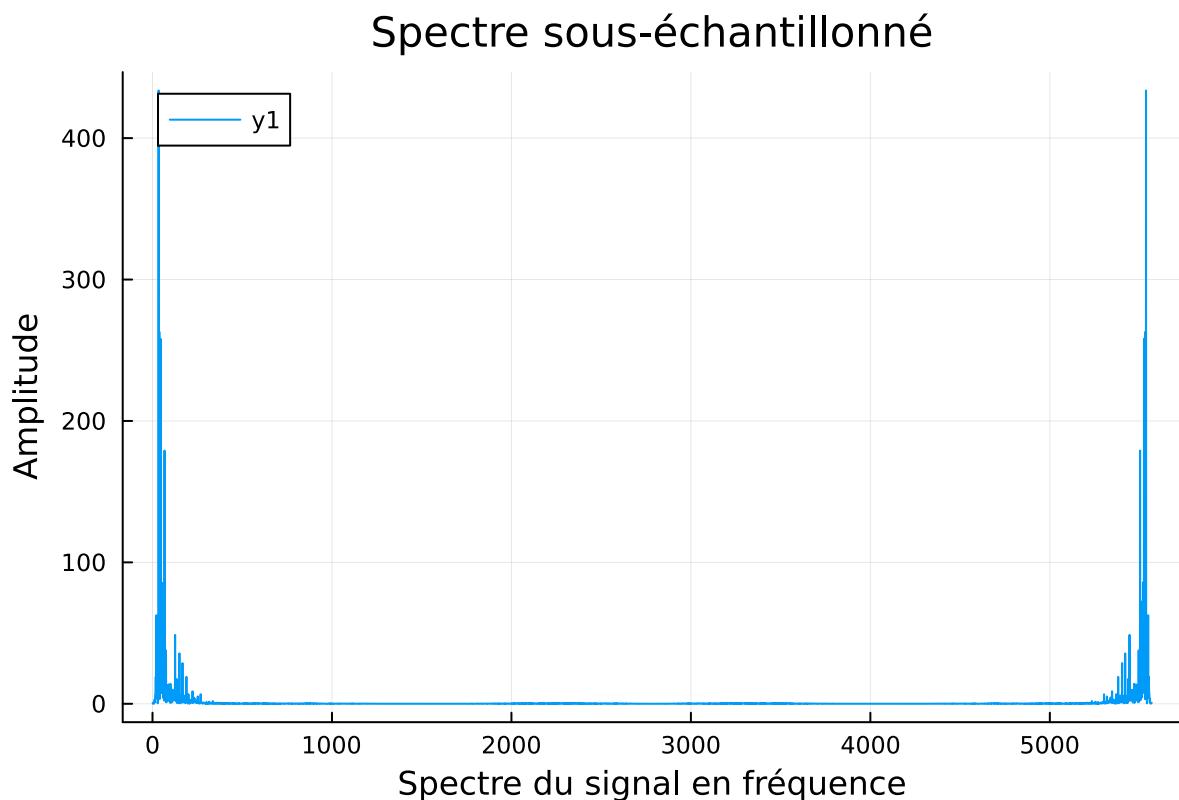
```
1 # Affichage du graphique (t, y(t))
2 begin
3     gamme_compressed = gamme[1:factor_gamme:end]
4     t_gamme = time_vector(gamme, gamme_compressed, fs_gamme)
5     plot(t_gamme, gamme_compressed, xlabel="Temps (s)", ylabel="Amplitude",
6           title="Signal de la Gamme Do3 à Do4", legend=false)
6 end
```

2.2. Affichage du spectre du signal

Pour afficher le spectre $Y(i\omega)$ du signal $y(t)$, on utilise la fonction `fft` (**Fast Fourier Transform**) de la bibliothèque `FFTW` en Julia.

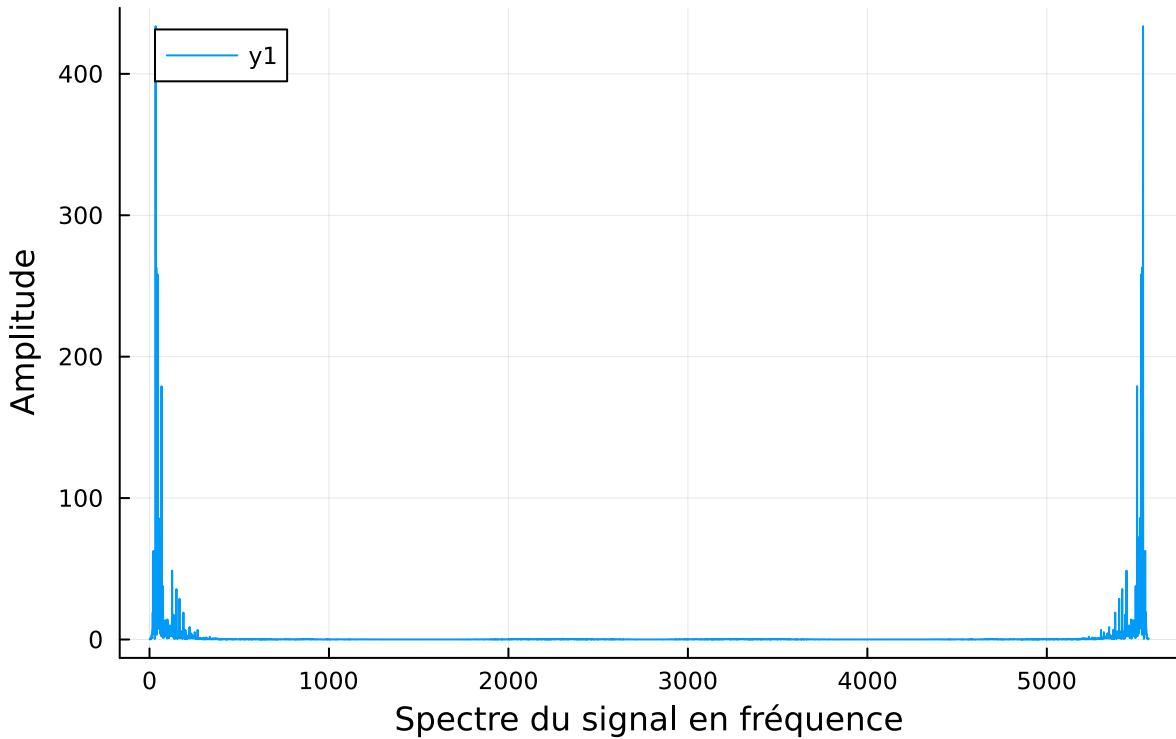
Notez que cette fonction renvoie initialement le spectre dans l'intervalle fréquentiel correspondant à la première période du signal, c'est-à-dire entre $[0, 2\pi]$. La fonction `fftshift` permet de recentrer la transformée de Fourier pour qu'elle soit centrée autour de zéro, dans l'intervalle $[-\pi, \pi]$.

- 1. Affichez le spectre du signal dans l'intervalle $[-\pi, \pi]$. N'oubliez pas de calculer le vecteur des fréquences correspondantes, qui doit être de la même taille que le spectre.
- 2. Affichez le même spectre, mais cette fois dans l'intervalle $[-\frac{f_s}{2}, \frac{f_s}{2}]$, où f_s est la fréquence d'échantillonnage du signal.



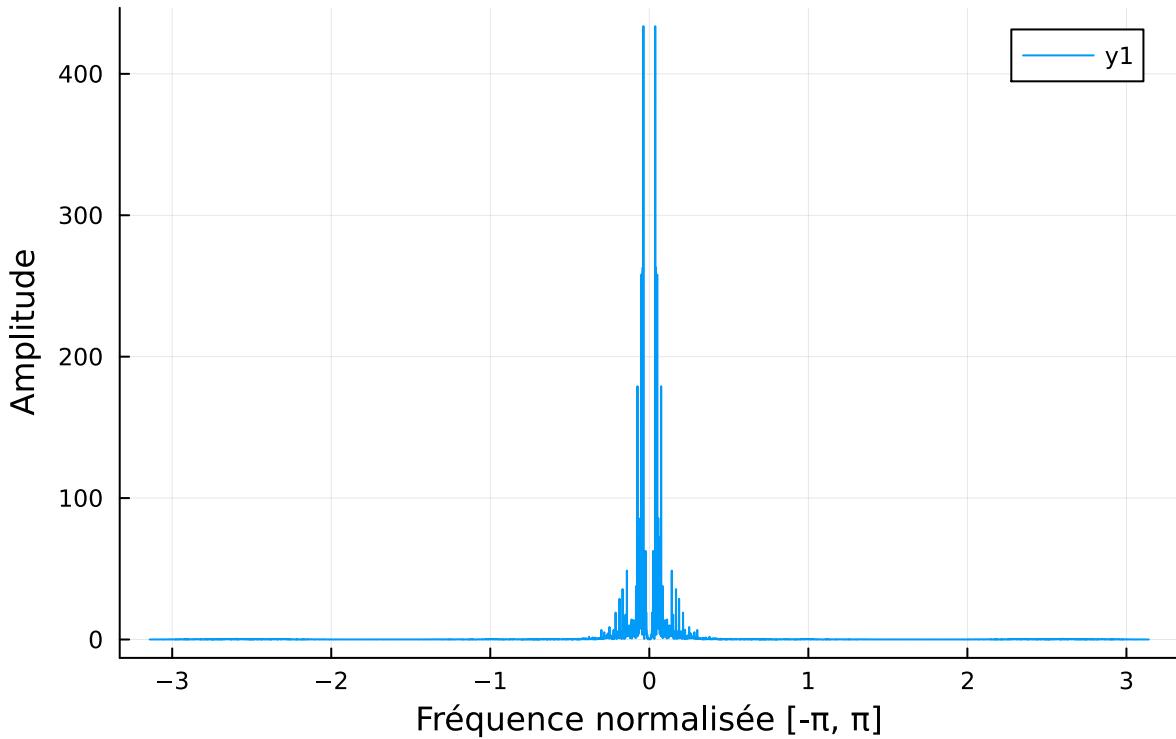
```
1 begin
2     # a. Transformée de Fourier Y(jw) de y(t) :
3     Y = fft(gamme)
4
5     plot(abs.(Y[1:factor_gamme:end]), xlabel="Spectre du signal en fréquence",
6         ylabel="Amplitude", title="Spectre sous-échantillonné")
6 end
```

Spectre sous-échantillonné nettoyé



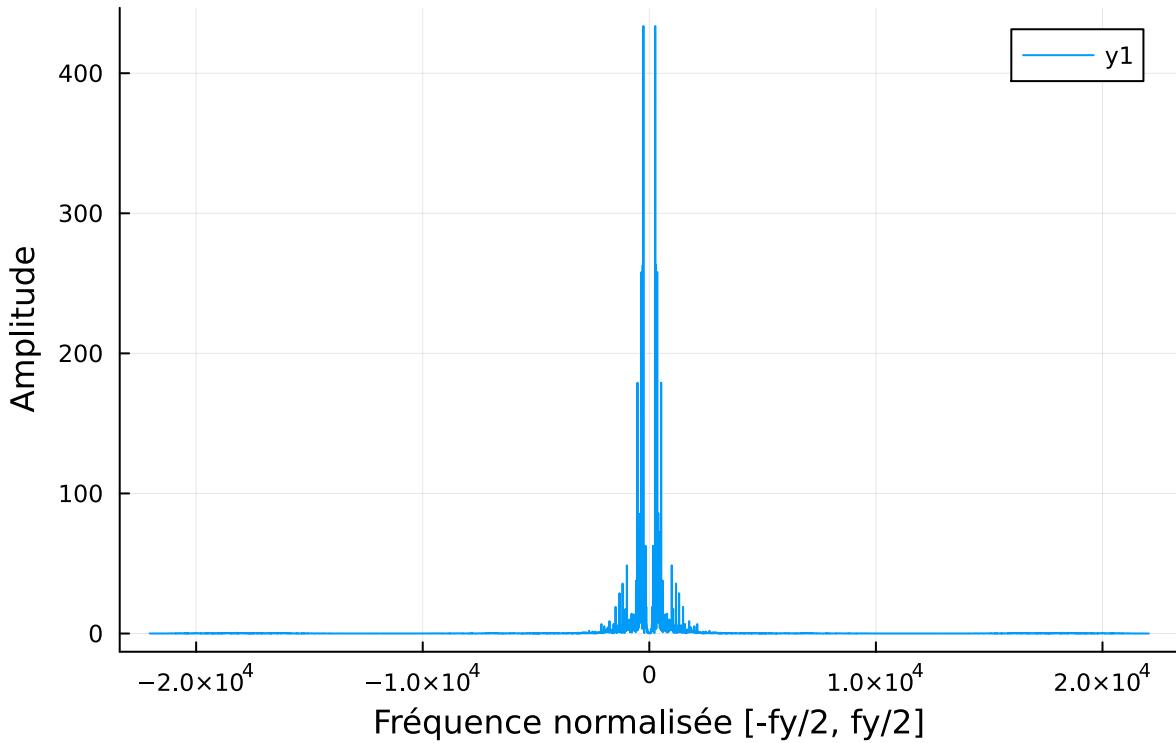
```
1 begin
2   # b. Nettoyage du spectre pour supprimer les valeurs du module plus petites que
3   # 1e-4 sont mises à zéro
4
5   Y[abs.(Y) < 1e-4] .= 0.0
6
7   plot(abs.(Y[1:factor_gamma:end]), xlabel="Spectre du signal en fréquence",
8       ylabel="Amplitude", title="Spectre sous-échantillonné nettoyé")
9 end
```

Spectre sous-échantillonné entre $[-\pi, \pi]$



```
1 begin
2     # c. Affichage du spectre entre -pi et pi
3     Y_shifted = fftshift(Y)
4
5     # Calcul du vecteur des fréquences normalisées pour [-pi, pi]
6     omega = LinRange(-pi, pi, n_gamme)
7
8     # Sous-échantillonnage du spectre
9     Y_sub = Y_shifted[1:factor_gamme:end]
10    omega_sub = omega[1:factor_gamme:end]
11
12    # Affichage du spectre sous-échantillonné
13
14    plot(omega_sub, abs.(Y_sub), xlabel="Fréquence normalisée [-\pi, \pi]",
15          ylabel="Amplitude", title="Spectre sous-échantillonné entre [-\pi, \pi]")
16 end
```

Spectre sous-échantillonné entre [-fs/2, fs/2]



```
1 begin
2     # d. Affichage du spectre en -fy/2 et fy/2
3     omega2 = LinRange(-fs_gamme/2, fs_gamme/2, n_gamme)
4
5     # Sous-échantillonnage du spectre
6     Y_sub2 = Y_shifted[1:factor_gamme:end]
7     omega_sub2 = omega2[1:factor_gamme:end]
8
9     # Affichage du spectre sous-échantillonné
10
11    plot(omega_sub2, abs.(Y_sub2), xlabel="Fréquence normalisée [-fy/2, fy/2]", ylabel="Amplitude", title="Spectre sous-échantillonné entre [-fs/2, fs/2]")
12 end
```

Y a-t-il une différence entre ces deux représentations du spectre ?

On constate que le spectre est très étendu et que beaucoup de fréquences ne sont pas nécessaires. La fréquence d'échantillonnage de base est donc très rapide : on peut se permettre de la réduire.

2.3. Retrouver le signal

Admettons qu'on n'ait accès qu'à ce spectre et qu'on souhaite retrouver le signal initial. Pour cela, on va utiliser la transformée de Fourier inverse `ifft` pour Inverse Fast Fourier Transform).

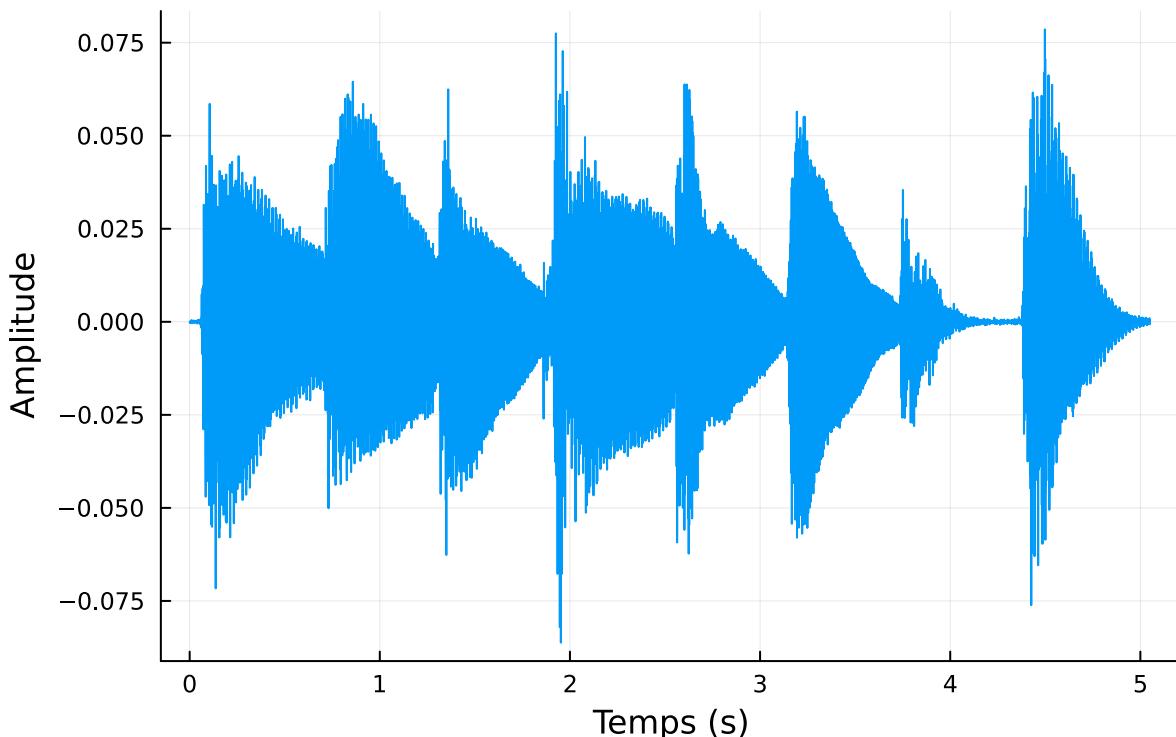
Retrouvez le signal $ys(t)$ récupéré depuis le spectre $Y(i\omega)$ du signal initial.

Astuce : attention, votre spectre a été shifté sur l'intervalle $[-\pi, \pi]$. Pensez à le ramener à sa bonne place avec `ifftshift`. N'oubliez pas non plus de ne garder que la partie réelle du signal, car des coefficients complexes peuvent traîner à cause des erreurs numériques...

```
[-0.000213623, 0.000366211, -3.05176e-5, -6.10352e-5, 0.000152588, -9.15527e-5, 9.15527e-
```

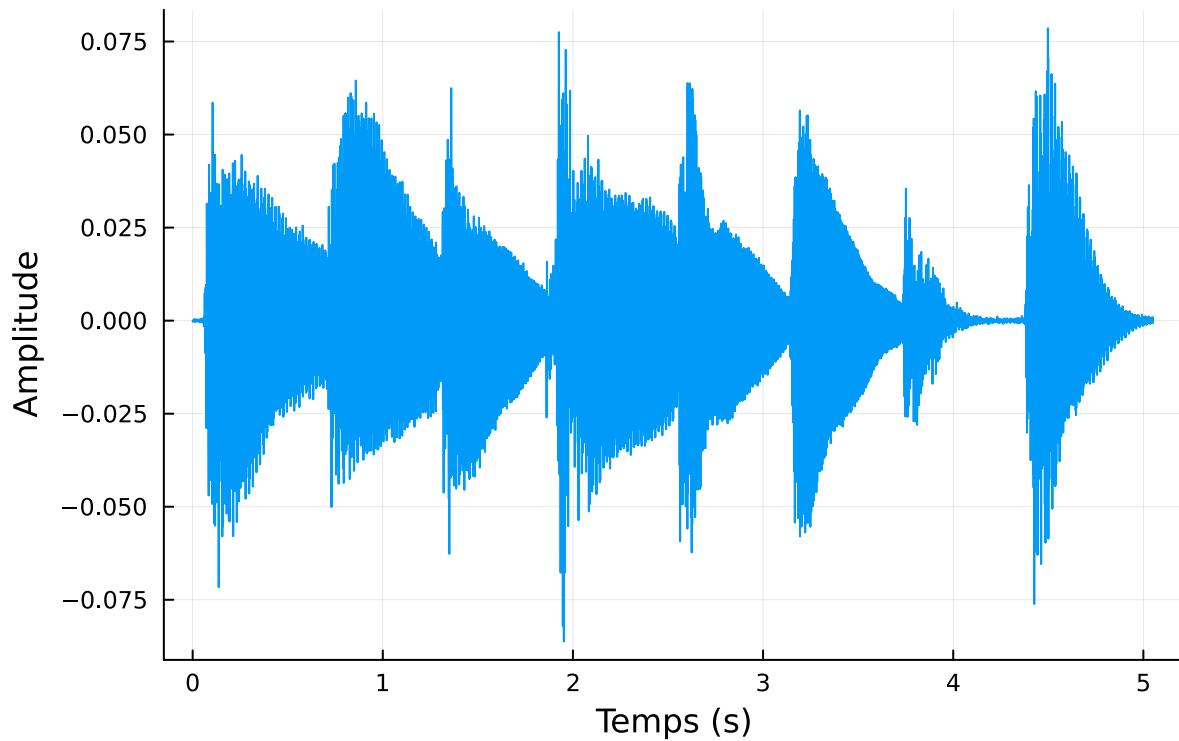
```
1 # a. Transformée de Fourier Inverse après replacement du spectre en [0, 2pi]
2 begin
3     # Remettre le spectre à sa position originale avec ifftshift
4     Y_unshifted = ifftshift(Y_shifted)
5
6     # Appliquer la transformée de Fourier inverse pour récupérer le signal temporel
7     ys = real(ifft(Y_unshifted))
8 end
```

Signal original

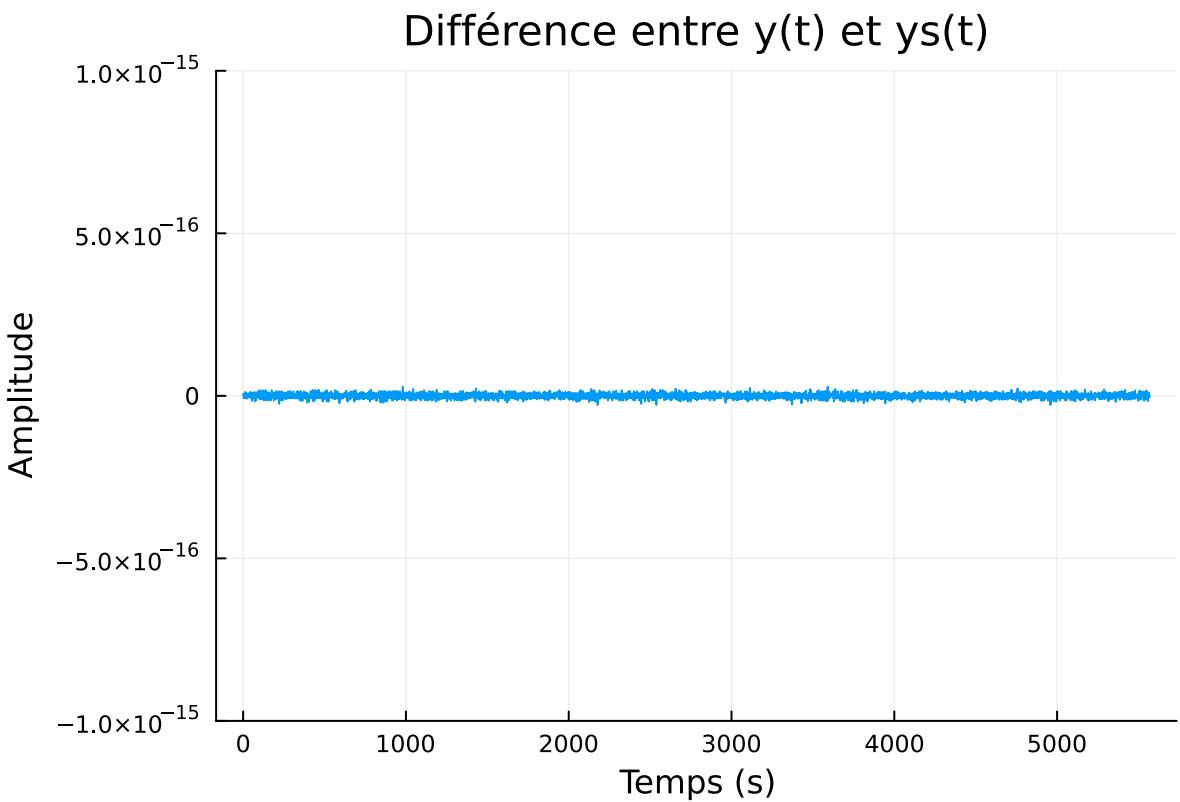


```
1 # b. Affichage du signal original y
2 plot(t_gamme, gamme[1:factor_gamme:end], title="Signal original", xlabel="Temps
(s)", ylabel="Amplitude", legend=false)
```

Signal recomposé



```
1 # c. Affichage du signal recomposé ys
2 plot(t_gamma, ys[1:factor_gamma:end], title="Signal recomposé", xlabel="Temps (s)",
       ylabel="Amplitude", legend=false)
```



```

1 # d. Calcul et affichage de la différence entre le signal original et le signal
  recomposé
2 begin
3   difference = gamme - ys
4
5   # Affichage de la différence de reconstruction
6   plot(difference[1:factor_gamme:end], title="Différence entre y(t) et ys(t)",
9      xlabel="Temps (s)", ylabel="Amplitude", legend=false, ylim=(-1e-15, 1e-15))
7 end

```

222720

```

1 #e. Jouez le son du signal recomposé
2 PortAudioStream() do s
3   write(s, ys)
4 end

```

3. Échantillonnage de signaux

Nous allons sous-échantillonner fortement un signal et observer l'effet sur le signal temporel. Ensuite, nous l'échantillonnerons à une fréquence $f_e = f_{\max}$ qui pourrait paraître suffisante dans le domaine temporel mais qui va mener à du repli spectral lorsqu'on fait un aller-retour dans le domaine de Fourier (par exemple pour filtrer son contenu).

3.1. Crédit à la création signal

Créer un signal sinusoïdal $z(t) = \sin(2\pi ft)$ échantillonné à 44100 Hz, de fréquence $f = 440$ Hz, et durant 0.1 secondes. Écouter le signal et dessiner sa réponse temporelle.

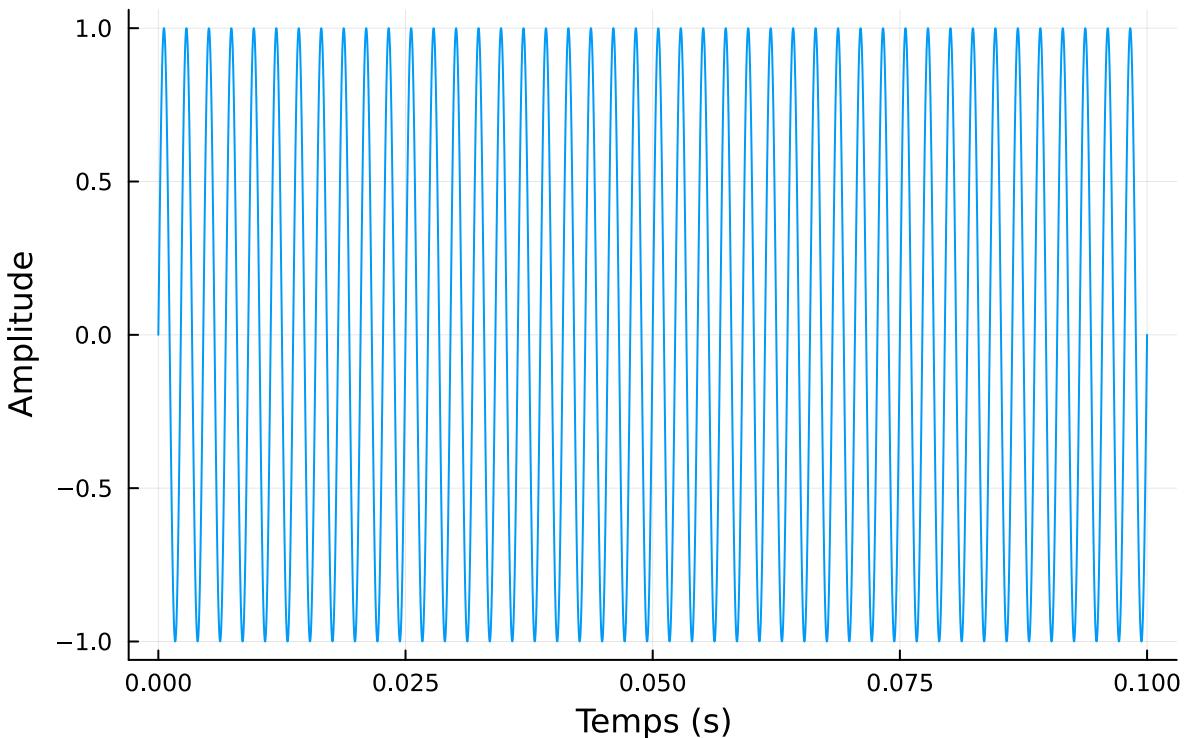
```
[0.0, 0.0626625, 0.125079, 0.187003, 0.248193, 0.308407, 0.367409, 0.424967, 0.480854, 0.!
```

```
1 begin
2     fs = 44100 # Fréquence d'échantillonnage en Hz
3     f = 440 # Fréquence de la sinusoïde en Hz
4     duration = 0.1 # Durée du signal en secondes
5
6     # Générer le vecteur de temps
7     t = LinRange(0, duration, floor(Int, fs * duration))
8
9     # Générer le signal sinusoïdal z(t) = sin(2πft)
10    z = sin.(2π * f * t)
11 end
```

4410

```
1 # a. Écouter le signal
2 PortAudioStream() do stream
3     write(stream, z)
4 end
```

Réponse temporelle de $z(t)$



```
1 # b. Afficher la réponse temporelle
2 plot(t, z, xlabel="Temps (s)", ylabel="Amplitude", title="Réponse temporelle de
z(t)", legend=false)
```

3.2. Echantillonage

Créer le vecteur z_{e1} et z_{e2} en échantillonnant le sinus à respectivement 441 Hz et 882 Hz. Représenter les échantillons sur un même graphique reprenant $z(t)$, $z_{e1}(t)$ et $z_{e2}(t)$. Que constatez-vous ?

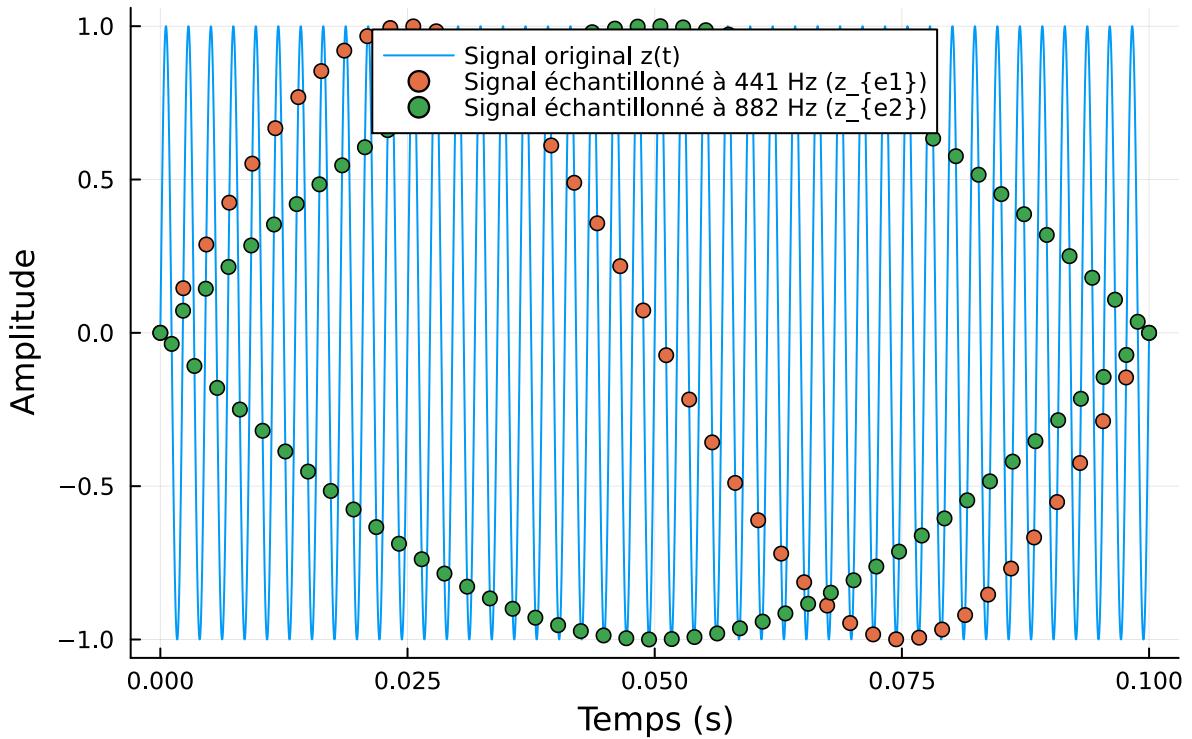
```
[0.0, 0.145601, 0.288099, 0.424457, 0.551768, 0.667319, 0.768647, 0.853593, 0.920346, 0.96]
```

```
1 begin
2     # Échantillonage à 441 Hz
3     fs_e1 = 441 # Fréquence d'échantillonage réduite à 441 Hz
4     t_e1 = LinRange(0, duration, floor(Int, fs_e1 * duration))
5     z_e1 = sin.(2π * f * t_e1) # Signal échantilloné à 441 Hz
6 end
```

```
[0.0, -0.0361024, 0.0721578, -0.108119, 0.143939, -0.179572, 0.21497, -0.250089, 0.284881]
```

```
1 begin
2     # Échantillonage à 882 Hz
3     fs_e2 = 882 # Fréquence d'échantillonage réduite à 882 Hz
4     t_e2 = LinRange(0, duration, floor(Int, fs_e2 * duration))
5     z_e2 = sin.(2π * f * t_e2) # Signal échantilloné à 882 Hz
6 end
```

Comparaison des signaux échantillonnés



```
1 begin
2     # Représenter les trois signaux sur un même graphique
3     plot(t, z, label="Signal original z(t)", xlabel="Temps (s)",
4           ylabel="Amplitude", title="Comparaison des signaux échantillonnés", legend=:top)
5     scatter!(t_e1, z_e1, label="Signal échantillonné à 441 Hz (z_{e1})",
6               markersize=4)
7     scatter!(t_e2, z_e2, label="Signal échantillonné à 882 Hz (z_{e2})",
8               markersize=4)
9 end
```

3.3. Spectres

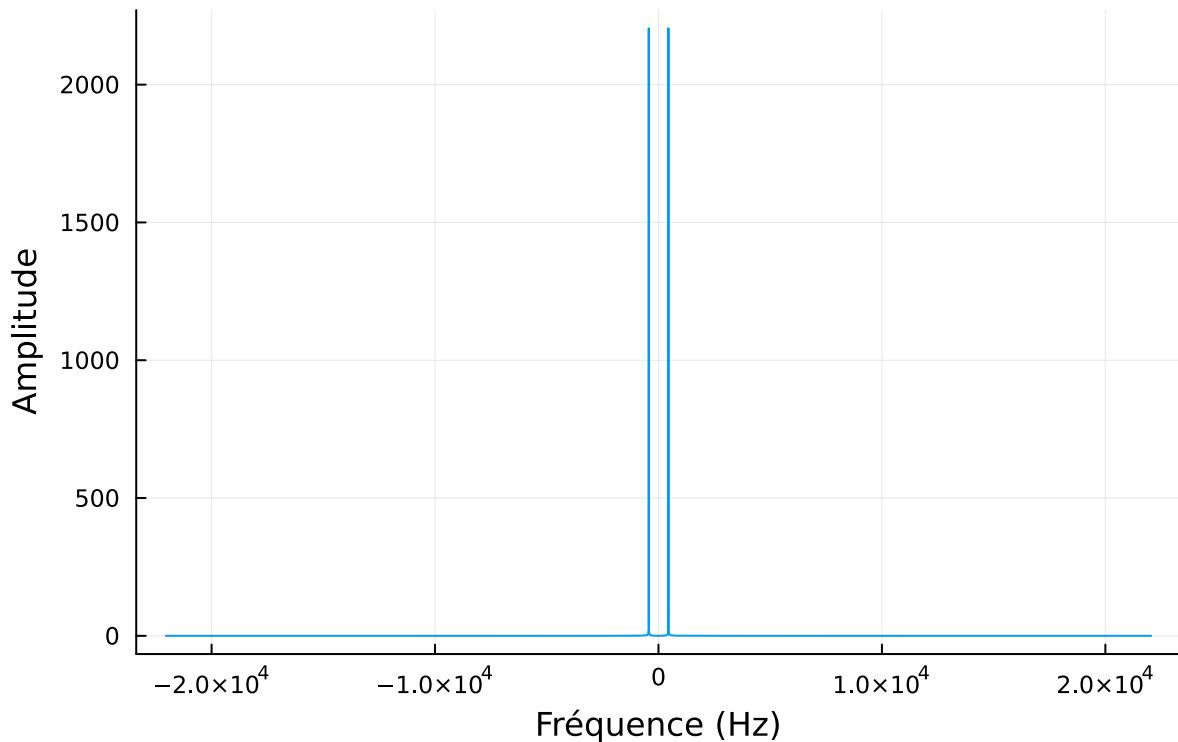
Représentez les trois spectres (module de la T.F.) sur trois subplots en prenant soin de graduer les abscisses en fréquence en Hertz. Que constatez-vous ? Quelles sont les fréquences fondamentales de chaque signal ? Sont-elles cohérentes avec le signal initial joué à 440 Hz ? Laquelle respecte le théorème de Shannon-Nyquist ?

Astuce : n'oubliez pas le fftshift !

```
[55.3799, 19.0294, 3.77997, 1.61683, 0.896621, 0.569386, 0.393211, 0.287509, 0.219109, 0.19109, 0.16208, 0.13207, 0.10206, 0.07205, 0.04204, 0.01203, -0.01796, -0.04797, -0.07798, -0.10799, -0.13799, -0.16799, -0.19799, -0.22799, -0.25799, -0.28799, -0.31799, -0.34799, -0.37799, -0.40799, -0.43799, -0.46799, -0.49799, -0.52799, -0.55799, -0.58799, -0.61799, -0.64799, -0.67799, -0.70799, -0.73799, -0.76799, -0.79799, -0.82799, -0.85799, -0.88799, -0.91799, -0.94799, -0.97799, -0.10206, -0.13207, -0.16208, -0.19209, -0.22209, -0.25209, -0.28209, -0.31209, -0.34209, -0.37209, -0.40209, -0.43209, -0.46209, -0.49209, -0.52209, -0.55209, -0.58209, -0.61209, -0.64209, -0.67209, -0.70209, -0.73209, -0.76209, -0.79209, -0.82209, -0.85209, -0.88209, -0.91209, -0.94209, -0.97209, 0.01203, 0.04204, 0.07205, 0.10206, 0.13207, 0.16208, 0.19209, 0.22209, 0.25209, 0.28209, 0.31209, 0.34209, 0.37209, 0.40209, 0.43209, 0.46209, 0.49209, 0.52209, 0.55209, 0.58209, 0.61209, 0.64209, 0.67209, 0.70209, 0.73209, 0.76209, 0.79209, 0.82209, 0.85209, 0.88209, 0.91209, 0.94209, 0.97209]
```

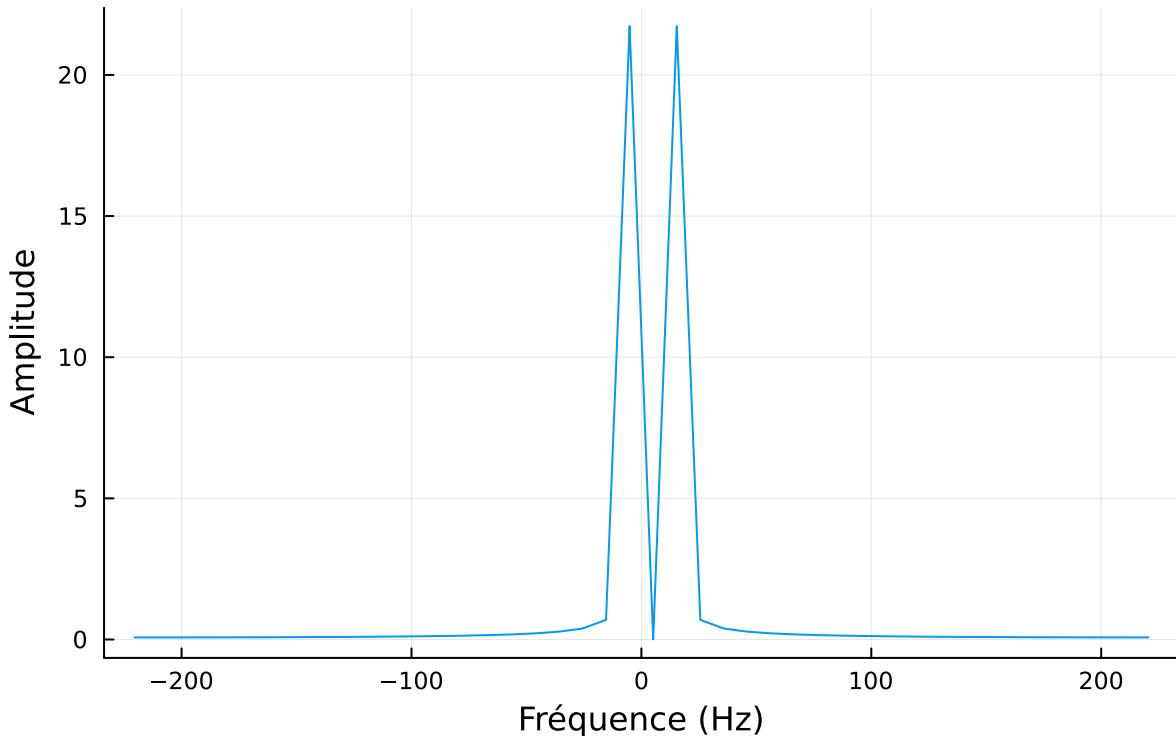
```
1 begin
2     # Calcul de la transformée de Fourier pour chaque signal et application de
3     # fftshift
4     Y3 = fftshift(fft(z))
5     Y_e1 = fftshift(fft(z_e1))
6     Y_e2 = fftshift(fft(z_e2))
7
8     # Convertir les fréquences en Hertz
9     frequencies = LinRange(-fs/2, fs/2, length(Y3))
10    frequencies_e1 = LinRange(-fs_e1/2, fs_e1/2, length(Y_e1))
11    frequencies_e2 = LinRange(-fs_e2/2, fs_e2/2, length(Y_e2))
12
13    # Calcul du module de la T.F. (norme)
14    Y_abs = abs.(Y3)
15    Y_e1_abs = abs.(Y_e1)
16    Y_e2_abs = abs.(Y_e2)
17
18 end
```

Spectre du signal original $z(t)$



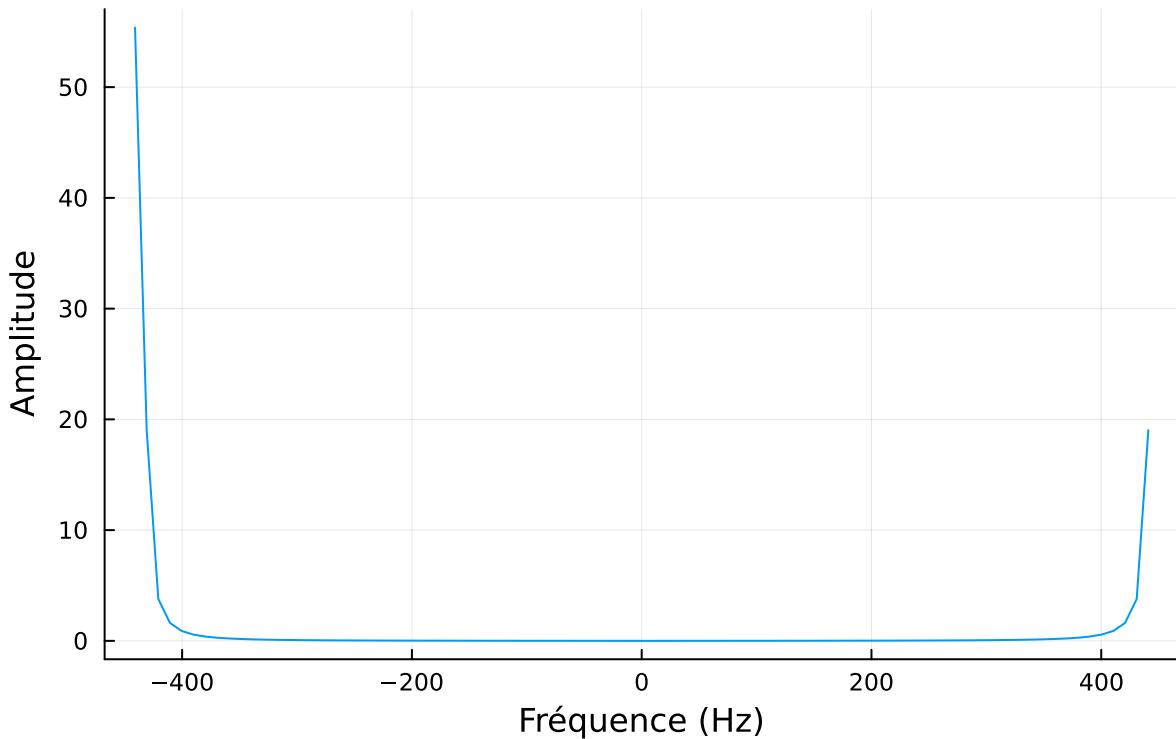
```
1 plot(frequencies, Y_abs, xlabel="Fréquence (Hz)", ylabel="Amplitude",
      title="Spectre du signal original z(t)", legend=false)
```

Spectre du signal échantillonné à 441 Hz



```
1 plot(frequencies_e1, Y_e1_abs, xlabel="Fréquence (Hz)", ylabel="Amplitude",
      title="Spectre du signal échantillonné à 441 Hz", legend=false)
```

Spectre du signal échantillonné à 882 Hz



```
1 plot(frequencies_e2, Y_e2_abs, xlabel="Fréquence (Hz)", ylabel="Amplitude",
      title="Spectre du signal échantillonné à 882 Hz", legend=false)
```

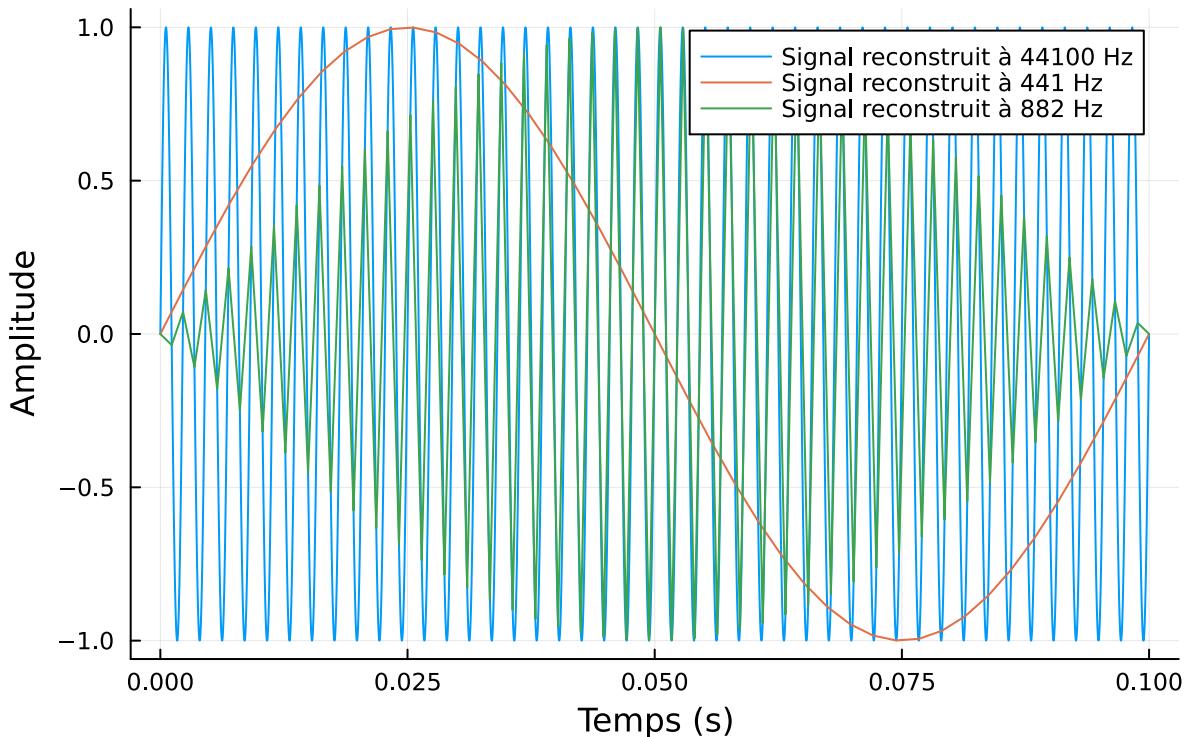
3.4. Reconstruction des signaux

Reconstruire les signaux à partir des trois spectres donnés et les superposer sur la même image.
Attention : le support n'est pas le même, puisque la fréquence d'échantillonnage est variable.

```
[1.61487e-16, -0.0361024, 0.0721578, -0.108119, 0.143939, -0.179572, 0.21497, -0.250089,
```

```
1 begin
2     # Reconstruction des signaux via la transformée de Fourier inverse
3     z_reconstruit = real(ifft(fftshift(Y3)))
4     z_e1_reconstruit = real(ifft(fftshift(Y_e1)))
5     z_e2_reconstruit = real(ifft(fftshift(Y_e2)))
6 end
```

Superposition des signaux reconstruits



```
1 begin
2     # Superposition des signaux reconstruits sur un même graphique
3     plot(t, z_reconstruit, label="Signal reconstruit à 44100 Hz", xlabel="Temps
(s)", ylabel="Amplitude", title="Superposition des signaux reconstruits")
4     plot!(t_e1, z_e1_reconstruit, label="Signal reconstruit à 441 Hz", markersize=4)
5     plot!(t_e2, z_e2_reconstruit, label="Signal reconstruit à 882 Hz", markersize=4)
6 end
```

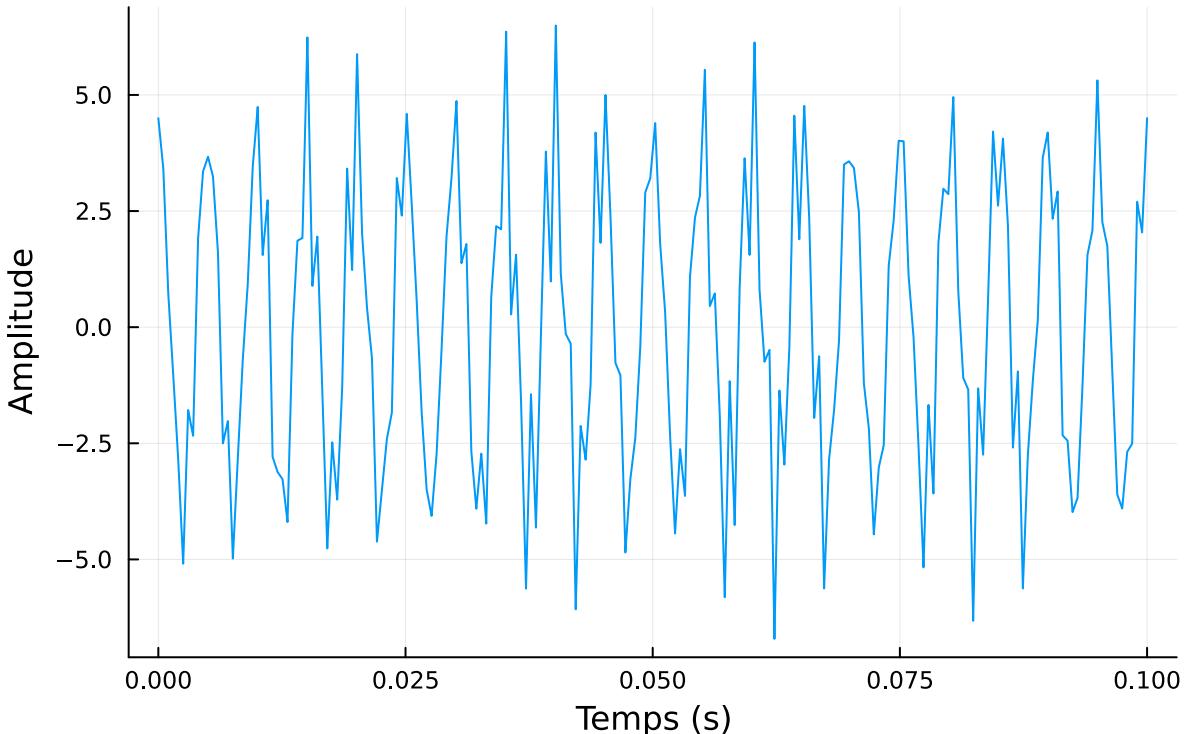
4. Filtrage d'un sinus bruité par des hautes fréquences

Dans cet exercice, on va tenter de séparer deux sinus mélangés dans le même signal. Considérez un signal $s(t) = 4\cos(2\pi ft)$, avec $f = 200$ Hz, d'une durée de 0.1 secondes, échantillonné à 2000 Hz.

```
[4.5, 3.41656, 0.723593, -1.03564, -2.84852, -5.09277, -1.78426, -2.33902, 1.90956, 3.354,
```

```
1 begin
2   fs4 = 2000
3   f4 = 200
4   duration4 = 0.1
5   t4 = LinRange(0, duration4, floor(Int, fs4 * duration4))
6   s_init = 4 * cos.(2π * f4 * t4)
7   z4 = s_init + .5 * cos.(10π * f4 * t4) + 1.1 * sin.(7.5π * f4 * t4) + .8 * sin.
8   (8π * f4 * t4) + .9 * sin.(12π * f4 * t4)
9 end
```

Signal dans le domaine temporel



```
1 plot(t4, z4, xlabel="Temps (s)", ylabel="Amplitude", title="Signal dans le domaine temporel", label="Signal", legend=false)
```

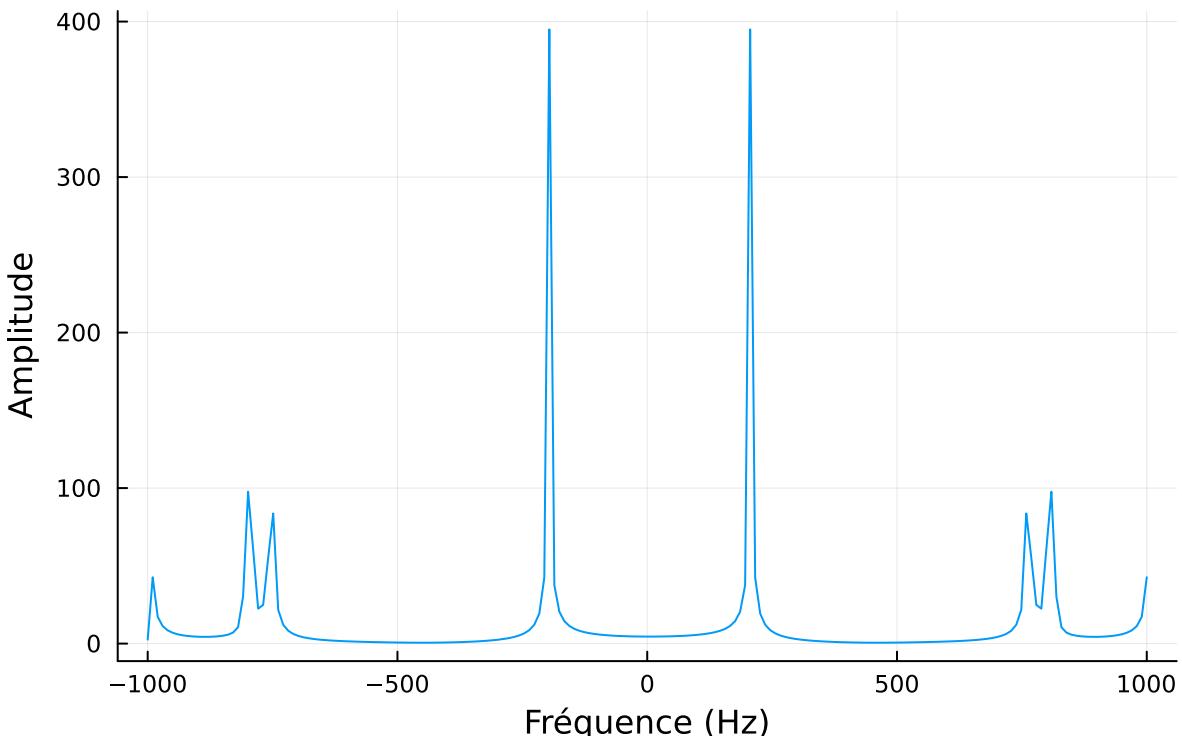
4.1. Transformée de Fourier

Calculer la transformée de Fourier , puis dessiner le signal et son spectre. Que constatez-vous sur ce spectre ?

```
[2.53368, 42.6532, 17.1674, 11.2002, 8.47517, 6.93217, 5.96224, 5.31894, 4.88416, 4.59553
```

```
1 begin
2     # Calcul de la transformée de Fourier pour chaque signal et application de
3     # fftshift
4     Y4 = fftshift(fft(z4))
5
6     # Convertir les fréquences en Hertz
7     frequencies4 = LinRange(-fs4/2, fs4/2, length(Y4))
8
9     # Calcul du module de la T.F. (norme)
10    Y_abs4 = abs.(Y4)
11 end
```

Spectre du signal



```
1 plot(frequencies4, Y_abs4, xlabel="Fréquence (Hz)", ylabel="Amplitude",
      title="Spectre du signal", legend=false)
```

4.2. Filtres

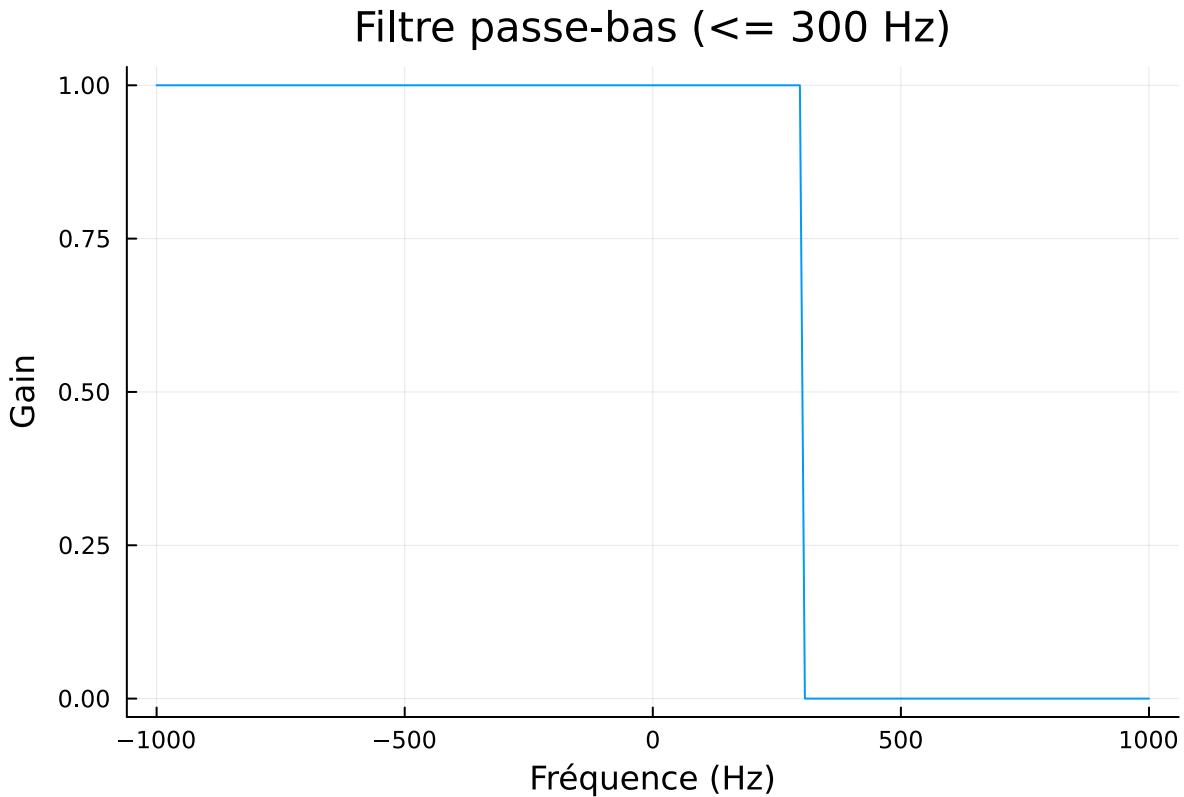
Réaliser un filtre passe bas limité à 300 Hz, ainsi qu'un filtre passe-bande laissant passer les fréquences entre 500 et 700 Hz. Représenter ces filtres.

Attention : vérifier que le support de vos filtres soit compatible avec votre spectre !

lowpass_filter (generic function with 1 method)

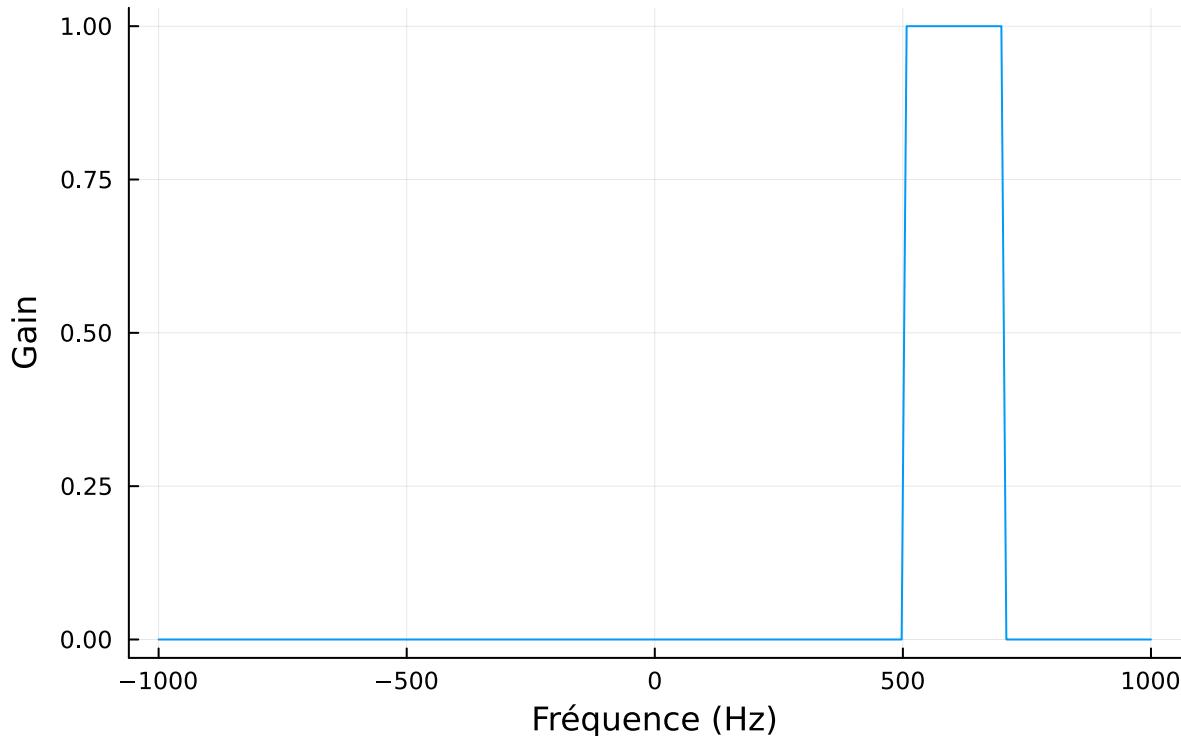
```
1 # a. Création du filtre passe-bas (limité à 300 Hz)
2 function lowpass_filter(f_limit, freq)
3     lowpass_filter = freq .<= f_limit
4     return lowpass_filter
5 end
```

```
bandpass_filter (generic function with 1 method)
1 # b. Création du filtre passe-bande (entre 500 et 700 Hz)
2 function bandpass_filter(f_low, f_high, freq)
3     bandpass_filter = (freq .>= f_low) .& (freq .<= f_high)
4     return bandpass_filter
5 end
```



```
1 plot(frequencies4, lowpass_filter(300, frequencies4), label="Filtre passe-bas",
      xlabel="Fréquence (Hz)", ylabel="Gain", title="Filtre passe-bas (<= 300 Hz)",
      legend=false)
```

Filtre passe-bande (500-700 Hz)



```
1 plot(frequencies4, bandpass_filter(500, 700, frequencies4), label="Filtre passe-bande", xlabel="Fréquence (Hz)", ylabel="Gain", title="Filtre passe-bande (500-700 Hz)", legend=false)
```

4.3. Ajout des filtres

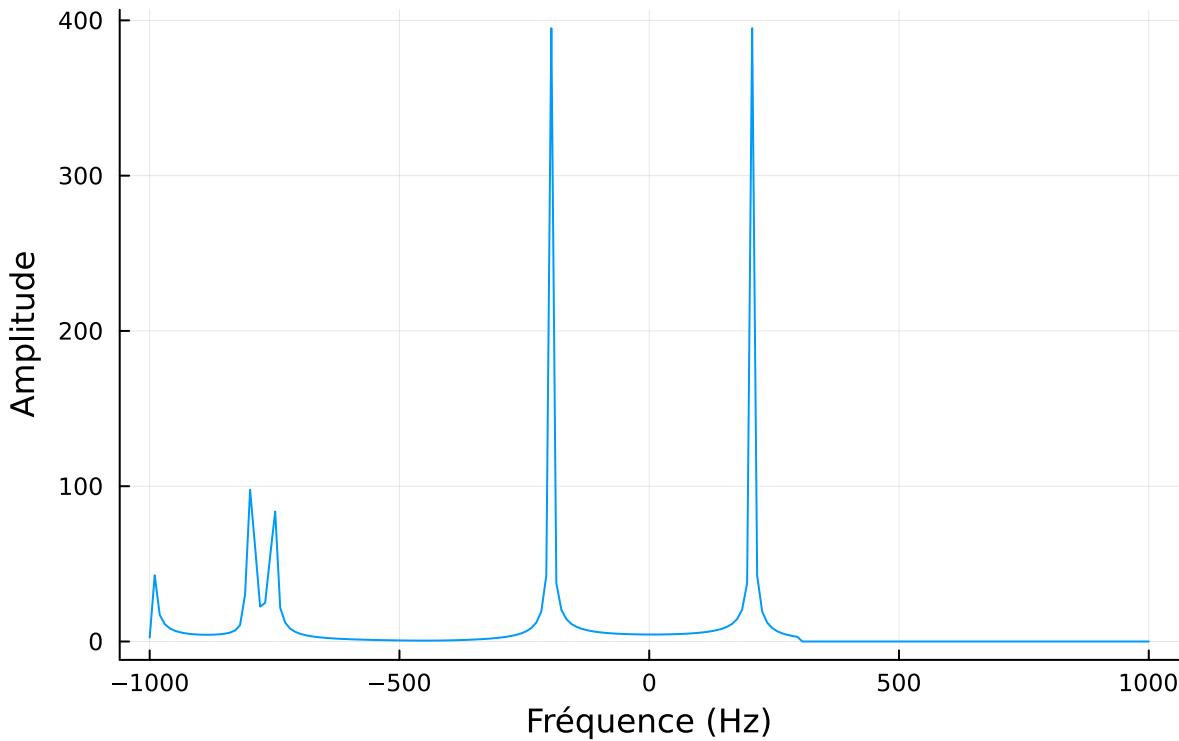
Multiplier les filtres avec le spectre $S(j\omega)$. Visualiser les spectres filtrés, puis représenter le signal retrouvé par transformée de Fourier inverse.

Astuce : ne pas oublier de shifter le spectre avant reconstruction !

```
[-0.0+0.0im, -0.0-0.0im, -0.0-0.0im, -0.0-0.0im, -0.0-0.0im, -0.0-0.0im, -0.0-0.0im, -0.0
```

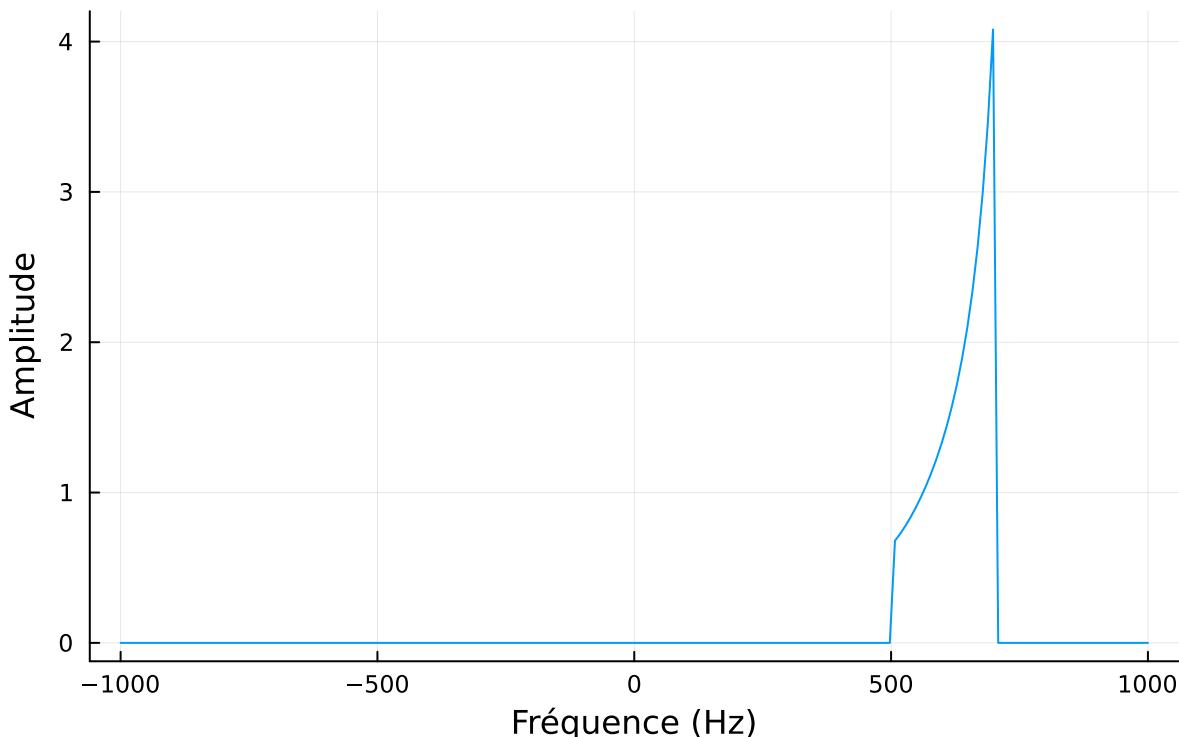
```
◀ ▶
1 # b. Multiplier les filtres avec le spectre
2 begin
3     S_lowpass = Y4 .* lowpass_filter(300, frequencies4)
4     S_bandpass = Y4 .* bandpass_filter(500, 700, frequencies4)
5 end
```

Spectre filtré par le passe-bas



```
1 # c. Visualisation des spectres filtrés  
2 plot(frequencies4, abs.(S_lowpass), xlabel="Fréquence (Hz)", ylabel="Amplitude",  
       title="Spectre filtré par le passe-bas", legend=false)
```

Spectre filtré par le passe-bande

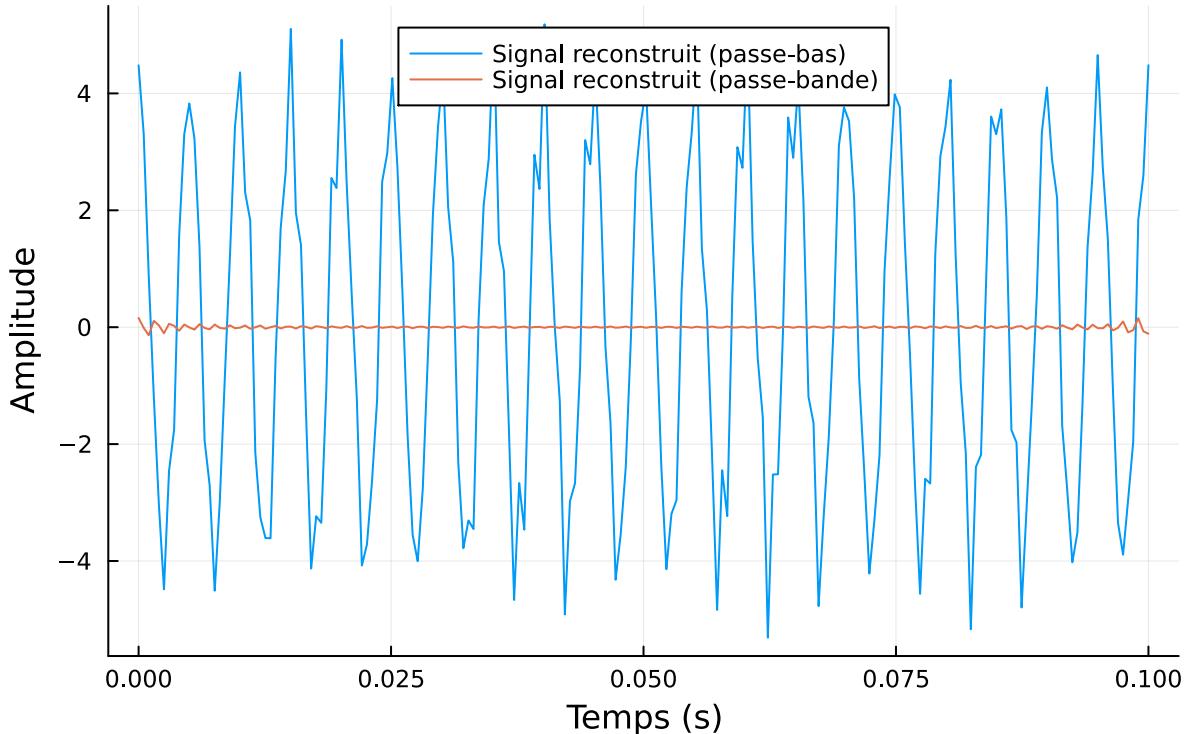


```
1 plot(frequencies4, abs.(S_bandpass), xlabel="Fréquence (Hz)", ylabel="Amplitude",  
       title="Spectre filtré par le passe-bande", legend=false)
```

```
[0.15875, -0.0100587, -0.137368, 0.108462, 0.0316808, -0.105336, 0.0583273, 0.0233729, -0
```

```
1 # d. Reconstruction des signaux par transformée de Fourier inverse
2 begin
3     s_lowpass_reconstructed = real(ifft(ifftshift(S_lowpass)))
4     s_bandpass_reconstructed = real(ifft(ifftshift(S_bandpass)))
5 end
```

Signal reconstruit avec le filtre passe-bas



```
1 # e. Visualisation des signaux reconstruits dans le domaine temporel
2 begin
3     plot(t4, s_lowpass_reconstructed, label="Signal reconstruit (passe-bas)",
        xlabel="Temps (s)", ylabel="Amplitude", title="Signal reconstruit avec le
        filtre passe-bas", legend=:top)
4     plot!(t4, s_bandpass_reconstructed, label="Signal reconstruit (passe-bande)")
5 end
```